

Speicherplatz-Komplexität

Warum sollte uns die Ressource „Speicherplatz“ interessieren?

Um

- ▶ die Komplexität der Berechnung von **Gewinnstrategien** für viele nicht-triviale 2-Personen Spiele zu charakterisieren,
- ▶ „nicht-klassische“ Berechnungsarten wie **Randomisierung** oder **Quanten-Berechnungen** mit deterministischen Berechnungen zu vergleichen,
- ▶ die Klasse **parallelisierbarer Probleme** besser zu verstehen.
- ▶ die Komplexität der folgenden Probleme für **NFAs** zu klären:
 - (?) Akzeptiert ein NFA eine gegebene Eingabe?
 - (?) Sind zwei NFAs äquivalent?
 - (?) Minimiere einen NFA.
- ▶ die Komplexität des **Wortproblems für kontextfreie Sprachen** oder kontextsensitive Sprachen zu untersuchen,

Wie misst man Speicherplatz?

Eine **I-O Turingmaschine** M besitzt drei ein-dimensionale Bänder mit jeweils einem Kopf, wobei jeder Kopf in einem Schritt (höchstens) zur linken oder rechten Nachbarzelle der gegenwärtig besuchten Zelle wandern darf.

- 1 Das erste Band ist das **read-only Leseband**, das die Eingabe speichert.
- 2 Das zweite Band ist das **read-write Arbeitsband**, das aus $s(n)$ Zellen besteht.
- 3 Das dritte Band ist das **write-only Ausgabeband**. (Wir interpretieren das Drucken einer 0 (1), als das Verwerfen (Akzeptieren) der Eingabe).

Eine **I-O Turingmaschine** M besitzt drei ein-dimensionale Bänder mit jeweils einem Kopf, wobei jeder Kopf in einem Schritt (höchstens) zur linken oder rechten Nachbarzelle der gegenwärtig besuchten Zelle wandern darf.

- 1 Das erste Band ist das **read-only Leseband**, das die Eingabe speichert.
- 2 Das zweite Band ist das **read-write Arbeitsband**, das aus $s(n)$ Zellen besteht.
- 3 Das dritte Band ist das **write-only Ausgabeband**. (Wir interpretieren das Drucken einer 0 (1), als das Verwerfen (Akzeptieren) der Eingabe).

Die Anzahl aller während der Berechnung für Eingabe w besuchten Zellen des zweiten Bands bezeichnen wir mit $DSPACE_M(w)$.

Speicherplatz: Determinismus

Wir messen den Speicherplatz

M sei eine I-O Turingmaschine.

(a) Der Speicherplatzbedarf von M für Eingabelänge n ist

$$\text{DSPACE}_M(n) = \max\{\text{DSPACE}_M(w) \mid w \in \Sigma^n\}.$$

(b) Für $s : \mathbb{N} \rightarrow \mathbb{N}$ ist

$$\text{DSPACE}(s) = \{L \subseteq \Sigma^* \mid L(M) = L \text{ für eine I-O TM } M \text{ mit } \text{DSPACE}_M = \mathcal{O}(s)\}$$

die Klasse aller auf Platz $\mathcal{O}(s(n))$ lösbarer Entscheidungsprobleme.

Wir messen den Speicherplatz

M sei eine I-O Turingmaschine.

- (a) Der Speicherplatzbedarf von M für Eingabelänge n ist

$$\text{DSPACE}_M(n) = \max\{\text{DSPACE}_M(w) \mid w \in \Sigma^n\}.$$

- (b) Für $s : \mathbb{N} \rightarrow \mathbb{N}$ ist

$$\text{DSPACE}(s) = \{L \subseteq \Sigma^* \mid L(M) = L \text{ für eine I-O TM } M \text{ mit } \text{DSPACE}_M = \mathcal{O}(s)\}$$

die Klasse aller auf Platz $\mathcal{O}(s(n))$ lösbaren Entscheidungsprobleme.

- (c) Die Komplexitätsklasse DL besteht aus allen mit logarithmischem Speicherplatzbedarf erkennbaren Sprachen, also

$$\text{DL} = \text{DSPACE}(\log_2 n).$$

Wir messen den Speicherplatz

M sei eine I-O Turingmaschine.

- (a) Der Speicherplatzbedarf von M für Eingabelänge n ist

$$\text{DSPACE}_M(n) = \max\{\text{DSPACE}_M(w) \mid w \in \Sigma^n\}.$$

- (b) Für $s : \mathbb{N} \rightarrow \mathbb{N}$ ist

$$\text{DSPACE}(s) = \{L \subseteq \Sigma^* \mid L(M) = L \text{ für eine I-O TM } M \text{ mit } \text{DSPACE}_M = \mathcal{O}(s)\}$$

die Klasse aller auf Platz $\mathcal{O}(s(n))$ lösbaren Entscheidungsprobleme.

- (c) Die Komplexitätsklasse DL besteht aus allen mit logarithmischem Speicherplatzbedarf erkennbaren Sprachen, also

$$\text{DL} = \text{DSPACE}(\log_2 n).$$

- (d) Die Komplexitätsklasse PSPACE besteht aus allen mit polynomielltem Speicherplatzbedarf erkennbaren Sprachen, also

$$\text{PSPACE} = \bigcup_{k \in \mathbb{N}} \text{DSPACE}(n^k).$$

Speicherplatz: Nichtdeterminismus

Speicherplatz nichtdeterministischer Rechnungen

Für eine nichtdeterministische TM M und eine Eingabe w ist $\text{NSPACE}_M(w)$ der maximale Speicherplatz einer Berechnung von M auf w .

(a) Der Speicherplatzbedarf von M für Eingabelänge n ist

$$\text{NSPACE}_M(n) = \max\{\text{NSPACE}_M(w) \mid w \in \Sigma^n\}.$$

Speicherplatz nichtdeterministischer Rechnungen

Für eine nichtdeterministische TM M und eine Eingabe w ist $\text{NSPACE}_M(w)$ der maximale Speicherplatz einer Berechnung von M auf w .

(a) Der Speicherplatzbedarf von M für Eingabelänge n ist

$$\text{NSPACE}_M(n) = \max\{\text{NSPACE}_M(w) \mid w \in \Sigma^n\}.$$

(b) Die Klasse aller mit Speicherplatz $\mathcal{O}(s)$ lösbaren Probleme ist

$$\text{NSPACE}(s) = \{L \subseteq \Sigma^* \mid \text{es gibt eine nichtdeterministische I-O TM } M \text{ mit } L(M) = L \text{ und } \text{NSPACE}_M = \mathcal{O}(s)\}.$$

Speicherplatz nichtdeterministischer Rechnungen

Für eine nichtdeterministische TM M und eine Eingabe w ist $\text{NSPACE}_M(w)$ der maximale Speicherplatz einer Berechnung von M auf w .

(a) Der Speicherplatzbedarf von M für Eingabelänge n ist

$$\text{NSPACE}_M(n) = \max\{\text{NSPACE}_M(w) \mid w \in \Sigma^n\}.$$

(b) Die Klasse aller mit Speicherplatz $\mathcal{O}(s)$ lösbaren Probleme ist

$$\text{NSPACE}(s) = \{L \subseteq \Sigma^* \mid \text{es gibt eine nichtdeterministische I-O TM } M \text{ mit } L(M) = L \text{ und } \text{NSPACE}_M = \mathcal{O}(s)\}.$$

(c) Die Komplexitätsklasse NL besteht aus allen nichtdeterministisch mit logarithmischem Speicherplatzbedarf erkennbaren Sprachen, also

$$\text{NL} = \text{NSPACE}(\log_2 n).$$

Speicherplatz nichtdeterministischer Rechnungen

Für eine nichtdeterministische TM M und eine Eingabe w ist $\text{NSPACE}_M(w)$ der maximale Speicherplatz einer Berechnung von M auf w .

- (a) Der Speicherplatzbedarf von M für Eingabelänge n ist

$$\text{NSPACE}_M(n) = \max\{\text{NSPACE}_M(w) \mid w \in \Sigma^n\}.$$

- (b) Die Klasse aller mit Speicherplatz $\mathcal{O}(s)$ lösbaren Probleme ist

$$\text{NSPACE}(s) = \{L \subseteq \Sigma^* \mid \text{es gibt eine nichtdeterministische I-O TM } M \text{ mit } L(M) = L \text{ und } \text{NSPACE}_M = \mathcal{O}(s)\}.$$

- (c) Die Komplexitätsklasse NL besteht aus allen nichtdeterministisch mit logarithmischem Speicherplatzbedarf erkennbaren Sprachen, also

$$\text{NL} = \text{NSPACE}(\log_2 n).$$

- (d) Die Komplexitätsklasse NPSpace besteht aus allen nichtdeterministisch, mit polynomielltem Speicherplatzbedarf erkennbaren Sprachen, also

$$\text{NPSpace} = \bigcup_{k \in \mathbb{N}} \text{NSPACE}(n^k).$$

Speicherplatz: Alternation

Speicherplatz alternierender Berechnungen

Der maximale Speicherplatzbedarf einer Berechnung eines alternierenden Algorithmus A auf Eingabe x ist

$$\text{aspace}_A(x).$$

(a) Der Speicherplatz von A für Eingabelänge n ist

$$\text{aspace}_A(n) := \max\{\text{aspace}_A(x) : x \in \Sigma^n\}$$

(b) Sei $s : \mathbb{N} \rightarrow \mathbb{N}$ gegeben. Dann definiere

$$\text{ASPACE}(s) := \{L \subseteq \Sigma^* : L = L(A) \text{ für eine alternierende I-O TM } A \text{ mit } \text{aspace}_A = \mathcal{O}(s)\}.$$

(c) Definiere „alternierenden polynomiellen Speicherplatz“ durch

$$\text{APSPACE} := \bigcup_{k \in \mathbb{N}} \text{ASPACE}(n^k).$$

Sind diese Klassen denn überhaupt interessant?

Sind diese Klassen überhaupt interessant? Und wie!

Es ist

(a) $\text{ASPACE}(\log_2 n) =$

Sind diese Klassen überhaupt interessant? Und wie!

Es ist

(a) $\text{ASPACE}(\log_2 n) = \text{P}$,

(b) $\text{APSPACE} =$

Sind diese Klassen überhaupt interessant? Und wie!

Es ist

(a) $\text{ASPACE}(\log_2 n) = \text{P}$,

(b) $\text{APSPACE} = \text{EXP}$ und

(c) $\text{AP} =$

Sind diese Klassen überhaupt interessant? Und wie!

Es ist

- (a) $\text{ASPACE}(\log_2 n) = P$,
- (b) $\text{APSPACE} = \text{EXP}$ und
- (c) $\text{AP} = \text{PSPACE}$.

Siehe Übungsaufgaben.

Speicherkomplexität: Die wichtigen Fragestellungen

? Ist DL eine echte Teilmenge von NL ?

- (*) Wie sehen die für DL **schwierigsten** Probleme in NL aus?
- (*) Für welches kleinste s gilt

$$NL \subseteq DSPACE(s).$$

- (*) Wenn $L \in NL$, ist dann auch $\bar{L} \in NL$? D.h. ist NL **abgeschlossen unter Komplement**?

? Ist NL eine Teilmenge von P , und wenn ja, ist NL eine echte Teilmenge?

? Sind alle Sprachen in NL parallelisierbar?

Speicherkomplexität: Die wichtigen Fragestellungen

? Ist DL eine echte Teilmenge von NL ?

- (*) Wie sehen die für DL **schwierigsten** Probleme in NL aus?
- (*) Für welches kleinste s gilt

$$NL \subseteq DSPACE(s).$$

- (*) Wenn $L \in NL$, ist dann auch $\bar{L} \in NL$? D.h. ist NL **abgeschlossen unter Komplement**?

? Ist NL eine Teilmenge von P , und wenn ja, ist NL eine echte Teilmenge?

? Sind alle Sprachen in NL parallelisierbar?

? Ist P eine echte Teilmenge von $PSPACE$?

- (*) Wie sehen die für P **schwierigsten** Probleme in $PSPACE$ aus?
- (*) Ist $NP \subseteq PSPACE$? Kann man allgemeine probabilistische Berechnungen, die in polynomieller Zeit ablaufen, in $PSPACE$ simulieren?
- (*) Lassen sich Quantenberechnungen in $PSPACE$ simulieren?

Speicherkomplexität: Die wichtigen Fragestellungen

? Ist DL eine echte Teilmenge von NL ?

- (*) Wie sehen die für DL **schwierigsten** Probleme in NL aus?
- (*) Für welches kleinste s gilt

$$NL \subseteq DSPACE(s).$$

- (*) Wenn $L \in NL$, ist dann auch $\bar{L} \in NL$? D.h. ist NL **abgeschlossen unter Komplement**?

? Ist NL eine Teilmenge von P , und wenn ja, ist NL eine echte Teilmenge?

? Sind alle Sprachen in NL parallelisierbar?

? Ist P eine echte Teilmenge von $PSPACE$?

- (*) Wie sehen die für P **schwierigsten** Probleme in $PSPACE$ aus?
- (*) Ist $NP \subseteq PSPACE$? Kann man allgemeine probabilistische Berechnungen, die in polynomieller Zeit ablaufen, in $PSPACE$ simulieren?
- (*) Lassen sich Quantenberechnungen in $PSPACE$ simulieren?

? Wie ordnen sich die Klassen regulärer, kontextfreier und kontextsensitiver Sprachen in die Speicherplatzklassen ein?

- (*) Die Chomsky-Hierarchie.

Sub-Logarithmischer Speicherplatz

Sub-logarithmischer Speicher

Für Eingaben der Länge n wird **logarithmischer** Speicher $\mathcal{O}(\log_2 n)$ benötigt, um sich an eine Eingabeposition zu erinnern.

Und wenn nur Speicher $o(\log_2 n)$ zur Verfügung steht?

(a) Aus welchen Sprachen besteht die Komplexitätsklasse $\text{DSPACE}(0)$?

Sub-logarithmischer Speicher

Für Eingaben der Länge n wird **logarithmischer** Speicher $\mathcal{O}(\log_2 n)$ benötigt, um sich an eine Eingabeposition zu erinnern.

Und wenn nur Speicher $o(\log_2 n)$ zur Verfügung steht?

- (a) Aus welchen Sprachen besteht die Komplexitätsklasse $\text{DSPACE}(0)$?
- ▶ TMs ohne Speicher sind **Zwei-Weg** Automaten.

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

- (b) Leben **nicht-reguläre** Sprachen in $\text{DSPACE}(\log_2 \log_2 n)$?

Sub-logarithmischer Speicher

Für Eingaben der Länge n wird **logarithmischer** Speicher $\mathcal{O}(\log_2 n)$ benötigt, um sich an eine Eingabeposition zu erinnern.

Und wenn nur Speicher $o(\log_2 n)$ zur Verfügung steht?

- (a) Aus welchen Sprachen besteht die Komplexitätsklasse $\text{DSPACE}(0)$?
- ▶ TMs ohne Speicher sind **Zwei-Weg** Automaten.

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

- (b) Leben **nicht-reguläre** Sprachen in $\text{DSPACE}(\log_2 \log_2 n)$?

- ▶ $\text{bin}(i)$ sei die Binärdarstellung der Zahl i ohne führende Nullen.
- ▶ Für Eingabealphabet $\Sigma = \{0, 1, \$\}$ definiere

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

- ▶ $\text{BIN} \in \text{DSPACE}(\log_2 \log_2 n)$: $\text{bin}(i)$ besteht für $i \leq n$ aus $\leq \lceil \log_2 n \rceil$ Bits.

Sub-logarithmischer Speicher

Für Eingaben der Länge n wird **logarithmischer Speicher** $\mathcal{O}(\log_2 n)$ benötigt, um sich an eine Eingabeposition zu erinnern.

Und wenn nur Speicher $o(\log_2 n)$ zur Verfügung steht?

- (a) Aus welchen Sprachen besteht die Komplexitätsklasse $\text{DSPACE}(0)$?
- ▶ TMs ohne Speicher sind **Zwei-Weg Automaten**.

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

- (b) Leben **nicht-reguläre Sprachen** in $\text{DSPACE}(\log_2 \log_2 n)$?

- ▶ $\text{bin}(i)$ sei die Binärdarstellung der Zahl i ohne führende Nullen.
- ▶ Für Eingabealphabet $\Sigma = \{0, 1, \$\}$ definiere

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

- ▶ $\text{BIN} \in \text{DSPACE}(\log_2 \log_2 n)$: $\text{bin}(i)$ besteht für $i \leq n$ aus $\leq \lceil \log_2 n \rceil$ Bits.
- ▶ BIN ist nicht regulär.

$\text{DSPACE}(o(\log_2 \log_2 n)) =$ Die Klasse der regulären Sprachen.

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-Szlepcsenyi beobachten.
- Zusammenhang zur Berechenbarkeit in logarithmischer paralleler Zeit.

Wir beginnen mit deterministisch logarithmischem Platz.

Das Bandalphabet –solange konstant groß – kann beliebig gewählt werden.

- Statt nur ein Arbeitsband können wir uns **konstant viele Arbeitsbänder** (mit jeweils einem Lese-Schreibkopf) erlauben.
 - ▶ Wir können uns das Arbeitsband in Spuren unterteilt denken.
- Statt nur einen Lesekopf auf dem Eingabeband können wir uns **konstant viele Leseköpfe** erlauben.
 - ▶ Speichere die Position jedes Lesekopfes auf dem Arbeitsband
 - ▶ und benutze einen Zähler, um die Position zu erreichen.

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

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

1. Arbeite mit zwei Leseköpfen:
 - ▶ der erste Kopf bleibt auf der ersten Eingabeposition stehen,
 - ▶ der zweite Kopf läuft zur letzten Eingabeposition.
2. Die beiden Köpfe vergleichen alle Buchstabenpaare $(w_i, w_{|w|-i+1})$, indem sie aufeinander zulaufen.

DL und NL,

Was kann man auf logarithmischem Platz rechnen?

In DL liegen

- 1 alle **regulären** Sprachen,
- 2 **einige kontextfreien Sprachen** wie etwa
 - ▶ die Palindromsprache,
 - ▶ die Dycksprache mit beliebig vielen Klammertypen,
- 3 sogar **einige** kontextsensitive Sprachen wie etwa $\{a^n b^n c^n \mid n \in \mathbb{N}\}$,
- 4 die Addition und Multiplikation von Zahlen in Binärdarstellung (und damit auch die Auswertung von Polynomen)
- 5 die Matrizenmultiplikation,
- 6 sogar **U-REACHABILITY**, die Menge aller **ungerichteten** Graphen G , die einen Weg von Knoten 1 nach Knoten 2 besitzen,
 - ▶ Der Graph G werde durch seine Adjazenzmatrix repräsentiert.
 - ▶ Der Nachweis ist kompliziert.

In DL liegen

- 1 alle **regulären** Sprachen,
- 2 **einige kontextfreien Sprachen** wie etwa
 - ▶ die Palindromsprache,
 - ▶ die Dycksprache mit beliebig vielen Klammertypen,
- 3 sogar **einige** kontextsensitive Sprachen wie etwa $\{a^n b^n c^n \mid n \in \mathbb{N}\}$,
- 4 die Addition und Multiplikation von Zahlen in Binärdarstellung (und damit auch die Auswertung von Polynomen)
- 5 die Matrizenmultiplikation,
- 6 sogar **U-REACHABILITY**, die Menge aller **ungerichteten** Graphen G , die einen Weg von Knoten 1 nach Knoten 2 besitzen,
 - ▶ Der Graph G werde durch seine Adjazenzmatrix repräsentiert.
 - ▶ Der Nachweis ist kompliziert.

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

(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 $s(n) \geq \log_2 n$. Dann ist

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

(c) Es ist $DL \subseteq P$.

(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 $s(n) \geq \log_2 n$. Dann ist

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

(c) Es ist $\text{DL} \subseteq P$.

(*) Teil (b) folgt aus Teil (a), wenn wir die Annahme $s(n) \geq \log_2 n$ beachten.

(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 $s(n) \geq \log_2 n$. Dann ist

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

(c) Es ist $DL \subseteq P$.

(*) Teil (b) folgt aus Teil (a), wenn wir die Annahme $s(n) \geq \log_2 n$ beachten.

(*) Teil (c) folgt aus Teil (b).

(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 $s(n) \geq \log_2 n$. Dann ist

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

(c) Es ist $DL \subseteq P$.

(*) Teil (b) folgt aus Teil (a), wenn wir die Annahme $s(n) \geq \log_2 n$ beachten.

(*) Teil (c) folgt aus Teil (b).

(*) Idee für Teil (a): Bei zu langer Laufzeit sollte eine platz-beschränkte Maschine in eine Endlosschleife geraten!

Für eine I-O Turingmaschine besteht eine **Konfiguration** aus

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

Für eine I-O Turingmaschine besteht eine **Konfiguration** 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 eine I-O Turingmaschine besteht eine **Konfiguration** 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

Für eine I-O Turingmaschine besteht eine **Konfiguration** 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

$$\mathcal{O}(n \cdot s \cdot 2^{\mathcal{O}(s)}) =$$

Für eine I-O Turingmaschine besteht eine **Konfiguration** 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

$$\mathcal{O}(n \cdot s \cdot 2^{\mathcal{O}(s)}) = n \cdot 2^{\mathcal{O}(s)}$$

Konfigurationen.

Für eine I-O Turingmaschine besteht eine **Konfiguration** 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

$$\mathcal{O}(n \cdot s \cdot 2^{\mathcal{O}(s)}) = n \cdot 2^{\mathcal{O}(s)}$$

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 Maschine rät einen Weg von Knoten 1 nach 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 .

Tatsächlich sind das Wortproblem für NFAs und D-REACHABILITY gleich-schwer.
- **2-SAT**, das Erfüllbarkeitsproblem für KNF-Formeln mit höchstens zwei Literalen pro Klausel gehört zu NL . Gehört 2-SAT zu DL ?
- Es gibt **kontextfreie Sprachen**, die zu NL aber wahrscheinlich nicht DL gehören.
 - ▶ Gehören alle kontextfreien Sprachen zu NL ?

Ist DL eine echte Teilmenge von NL ?

LOGSPACE-Reduktionen

Ist D-REACHABILITY deterministisch mit logarithmischem Speicherplatz erkennbar?

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

Ist D-REACHABILITY deterministisch mit logarithmischem Speicherplatz erkennbar?

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

LOGSPACE-Reduktionen.

L ist LOGSPACE-reduzierbar auf K , geschrieben

$$L \leq_{\text{LOG}} K,$$

falls es eine (deterministische) I-O Turingmaschine M mit logarithmischem Speicherplatzbedarf gibt, so dass für alle Eingaben $w \in \Sigma_1^*$,

$$w \in L \iff M(w) \in K.$$

- (a) Die Sprache K heißt **NL-hart**, falls $L \leq_{\text{LOG}} K$ für alle Sprachen $L \in \text{NL}$.
- (b) Die Sprache K heißt **NL-vollständig**, wenn K NL-hart ist und $K \in \text{NL}$.

- (a) Die Sprache K heißt **NL-hart**, falls $L \leq_{\text{LOG}} K$ für alle Sprachen $L \in \text{NL}$.
- (b) Die Sprache K heißt **NL-vollständig**, wenn K NL-hart ist und $K \in \text{NL}$.

Die wesentlichen Eigenschaften der LOGSPACE-Reduktion:

- Wenn $L \leq_{\text{LOG}} K$ und wenn $K \in \text{DL}$, dann ist

- (a) Die Sprache K heißt **NL-hart**, falls $L \leq_{\text{LOG}} K$ für alle Sprachen $L \in \text{NL}$.
- (b) Die Sprache K heißt **NL-vollständig**, wenn K NL-hart ist und $K \in \text{NL}$.

Die wesentlichen Eigenschaften der LOGSPACE-Reduktion:

- Wenn $L \leq_{\text{LOG}} K$ und wenn $K \in \text{DL}$, dann ist $L \in \text{DL}$.
 - ▶ $\text{DL} = \text{NL}$, wenn irgendeine NL-vollständige Sprache in DL liegt.
Die Sprache K sei NL-vollständig. Dann sind äquivalent

$$K \in \text{DL} \iff \text{DL} = \text{NL}.$$

- (a) Die Sprache K heißt **NL-hart**, falls $L \leq_{\text{LOG}} K$ für alle Sprachen $L \in \text{NL}$.
- (b) Die Sprache K heißt **NL-vollständig**, wenn K NL-hart ist und $K \in \text{NL}$.

Die wesentlichen Eigenschaften der LOGSPACE-Reduktion:

- Wenn $L \leq_{\text{LOG}} K$ und wenn $K \in \text{DL}$, dann ist $L \in \text{DL}$.
 - ▶ $\text{DL} = \text{NL}$, wenn irgendeine NL-vollständige Sprache in DL liegt.
Die Sprache K sei NL-vollständig. Dann sind äquivalent

$$K \in \text{DL} \iff \text{DL} = \text{NL}.$$

- Wenn $M \leq_{\text{LOG}} K$ und $K \leq_{\text{LOG}} L$, dann ist

- (a) Die Sprache K heißt **NL-hart**, falls $L \leq_{\text{LOG}} K$ für alle Sprachen $L \in \text{NL}$.
- (b) Die Sprache K heißt **NL-vollständig**, wenn K NL-hart ist und $K \in \text{NL}$.

Die wesentlichen Eigenschaften der LOGSPACE-Reduktion:

- Wenn $L \leq_{\text{LOG}} K$ und wenn $K \in \text{DL}$, dann ist $L \in \text{DL}$.
 - ▶ $\text{DL} = \text{NL}$, wenn irgendeine NL-vollständige Sprache in DL liegt.
Die Sprache K sei NL-vollständig. Dann sind äquivalent

$$K \in \text{DL} \iff \text{DL} = \text{NL}.$$

- Wenn $M \leq_{\text{LOG}} K$ und $K \leq_{\text{LOG}} L$, dann ist $M \leq_{\text{LOG}} L$.
 - ▶ Wie weist man nach, dass eine Sprache L NL-vollständig ist?
Wenn K NL-hart ist und wenn $K \leq_{\text{LOG}} L$, dann ist auch L NL-hart.

D-REACHABILITY ist NL -vollständig

M sei eine *nichtdeterministische* Turingmaschine mit logarithmischem Speicherplatz. Wir definieren das zentrale Konzept des

Berechnungsgraphen $G_M(w)$ von M auf Eingabe w .

- ▶ 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.

M sei eine *nichtdeterministische* Turingmaschine mit logarithmischem Speicherplatz. Wir definieren das zentrale Konzept des

Berechnungsgraphen $G_M(w)$ von M auf Eingabe w .

- ▶ 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.

M sei eine *nichtdeterministische* Turingmaschine mit logarithmischem Speicherplatz. Wir definieren das zentrale Konzept des

Berechnungsgraphen $G_M(w)$ von M auf Eingabe w .

- ▶ 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.

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

- 1 D-REACHABILITY liegt offensichtlich in NL .
- 2 *Zeige*: $L \leq_{LOG} D-REACHABILITY$ für eine beliebige Sprache $L \in NL$.

- 1 D-REACHABILITY liegt offensichtlich in NL .
- 2 *Zeige*: $L \leq_{\text{LOG}} \text{D-REACHABILITY}$ für eine beliebige Sprache $L \in NL$.
 - ▶ Es ist $L = L(M)$ für eine nichtdeterministische I-O Turingmaschine M mit logarithmischem Speicher.

- 1 D-REACHABILITY liegt offensichtlich in NL .
- 2 *Zeige*: $L \leq_{LOG} D\text{-REACHABILITY}$ für eine beliebige Sprache $L \in NL$.
 - ▶ Es ist $L = L(M)$ für eine nichtdeterministische I-O Turingmaschine M mit logarithmischem Speicher.
 - ▶ Der Berechnungsgraph $G_M(w)$ kann von einer deterministischen I-O TM berechnet werden.

- 1 D-REACHABILITY liegt offensichtlich in NL .
- 2 *Zeige:* $L \leq_{\text{LOG}} \text{D-REACHABILITY}$ für eine beliebige Sprache $L \in NL$.
 - ▶ Es ist $L = L(M)$ für eine nichtdeterministische I-O Turingmaschine M mit logarithmischem Speicher.
 - ▶ Der Berechnungsgraph $G_M(w)$ kann von einer deterministischen I-O TM berechnet werden.
 - ★ Der Knoten der Anfangskonfiguration erhält den Namen 1.

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

$$w \in L \iff$$

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

$$w \in L \iff G_M(w) \in \text{D-REACHABILITY.}$$

Weitere NL -vollständige Sprachen

- 1 Das Leerheitsproblem für DFAs (Akzeptiert ein gegebener DFA irgendeine Eingabe?)
- 2 Die Simulation von NFAs (Akzeptiert ein gegebener NFA eine gegebene Eingabe?)
- 3 2-SAT
- 4 Das Wortproblem für bestimmte kontextfreie Sprachen.

Beweis: In den [Übungen](#).

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

Wie mächtig ist NL?

$$DL \subseteq NL \subseteq P \subseteq NP.$$

- Die Beziehungen $DL \subseteq NL$ sowie $P \subseteq NP$ sind offensichtlich.
- Es genügt der Nachweis von $NL \subseteq P$.

Wie mächtig ist NL?

$$DL \subseteq NL \subseteq P \subseteq NP.$$

- Die Beziehungen $DL \subseteq NL$ sowie $P \subseteq NP$ sind offensichtlich.
- Es genügt der Nachweis von $NL \subseteq P$.
 - ▶ Für eine beliebige Sprache L in NL gilt $L \leq_{\text{LOG}} \text{D-REACHABILITY}$.

Wie mächtig ist NL?

$$DL \subseteq NL \subseteq P \subseteq NP.$$

- Die Beziehungen $DL \subseteq NL$ sowie $P \subseteq NP$ sind offensichtlich.
- Es genügt der Nachweis von $NL \subseteq P$.
 - ▶ Für eine beliebige Sprache L in NL gilt $L \leq_{\text{LOG}} \text{D-REACHABILITY}$.
 - ▶ Wenn $L \leq_{\text{LOG}} K$ und $K \in P$, dann ist auch $L \in P$. Warum?

Wie mächtig ist NL?

$$DL \subseteq NL \subseteq P \subseteq NP.$$

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

Wie mächtig ist NL?

$$DL \subseteq NL \subseteq P \subseteq NP.$$

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

Die Chomsky-Hierarchie: Worum geht's?

Die Chomsky-Hierarchie

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

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

- 2 Eine Grammatik G mit Produktionen der Form

$$u \rightarrow v \quad \text{mit } |u| \leq |v|$$

heißt **Typ 1** oder **kontextsensitiv**. Die zugehörige Sprachenfamilie ist

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

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

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

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

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

Die Chomsky Hierarchie und Platzkomplexität

Es kann gezeigt werden:

- (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) Es ist $\text{NL} \subseteq \text{LOGCFL} \subseteq \text{DSPACE}(\log_2^2 n)$. In LOGCFL liegen alle Sprachen, die LOGSPACE-reduzierbar auf eine kontextfreie Sprache sind. Insbesondere ist $\mathcal{L}_2 \subseteq \text{DSPACE}(\log_2^2 n)$.
- (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

Typ-0 Grammatiken G

- $L(G)$ ist rekursiv aufzählbar, d.h. $L(G) = L(M)$ für eine TM M .

Typ-0 Grammatiken G

- $L(G)$ ist rekursiv aufzählbar, d.h. $L(G) = L(M)$ für eine TM M .
 - ▶ $w \in L(G) \iff S \xrightarrow{*} w$: 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?

Typ-0 Grammatiken G

- $L(G)$ ist rekursiv aufzählbar, d.h. $L(G) = L(M)$ für eine TM M .
 - ▶ $w \in L(G) \iff S \xrightarrow{*} w$: 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 $L = L(M)$ für eine Turingmaschine M .

Konstruiere eine Grammatik G mit $L(M) = L(G)$.

- ▶ Für Eingabe w : Berechnungen von M **beginnen** mit w , Ableitungen von w **enden** mit w .
- ▶ Die Grammatik G sollte Berechnungen von M „rückwärts“ simulieren?!

Typ-0 Grammatiken G

- $L(G)$ ist rekursiv aufzählbar, d.h. $L(G) = L(M)$ für eine TM M .
 - ▶ $w \in L(G) \iff S \xrightarrow{*} w$: 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 $L = L(M)$ für eine Turingmaschine M .

Konstruiere eine Grammatik G mit $L(M) = L(G)$.

- ▶ Für Eingabe w : Berechnungen von M **beginnen** mit w , Ableitungen von w **enden** mit w .
- ▶ Die Grammatik G sollte Berechnungen von M „rückwärts“ simulieren?!
 - ★ Die Grammatik G wird die Konfigurationen

$\alpha_1 \cdots \alpha_{j-1} q \alpha_j \cdots \alpha_N$ (Die Maschine befindet sich im Zustand q , hat den Bandinhalt $\alpha_1 \cdots \alpha_N$ erstellt und liest das j -te Symbol des Bandes)

in umgekehrter Reihenfolge konstruieren.

Typ-0 Grammatiken G

- $L(G)$ ist rekursiv aufzählbar, d.h. $L(G) = L(M)$ für eine TM M .
 - ▶ $w \in L(G) \iff S \xrightarrow{*} w$: 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 $L = L(M)$ für eine Turingmaschine M .

Konstruiere eine Grammatik G mit $L(M) = L(G)$.

- ▶ Für Eingabe w : Berechnungen von M **beginnen** mit w , Ableitungen von w **enden** mit w .
- ▶ Die Grammatik G sollte Berechnungen von M „rückwärts“ simulieren?!
 - ★ Die Grammatik G wird die Konfigurationen
$$\alpha_1 \cdots \alpha_{j-1} q \alpha_j \cdots \alpha_N$$
(Die Maschine befindet sich im Zustand q , hat den Bandinhalt $\alpha_1 \cdots \alpha_N$ erstellt und liest das i te Symbol des Bandes) in umgekehrter Reihenfolge konstruieren.
- ▶ Annahmen an M :
 - ★ M besitzt nur einen **akzeptierenden Zustand** q_a .
 - ★ Akzeptierende Berechnungen enden mit dem leeren Band.
 - ★ M nimmt den Zustand q_0 nur zu Beginn der Berechnung an.

$G := (\Sigma, V, S, P)$ mit der Variablenmenge $V := (\Gamma \setminus \Sigma) \cup Q \cup \{\epsilon\}$ und $S := q_a$.

- Erzeuge den von M benutzten Bandbereich durch $q_a \rightarrow Bq_a \mid q_aB$.
- Dann beginnt die Rückwärtsrechnung.
 - ▶ Wenn $\delta(q, a) = (q', b, \text{links})$, wähle für alle $c \in \Gamma$ die Produktion

$G := (\Sigma, V, S, P)$ mit der Variablenmenge $V := (\Gamma \setminus \Sigma) \cup Q \cup \{\epsilon\}$ und $S := q_a$.

- Erzeuge den von M benutzten Bandbereich durch $q_a \rightarrow Bq_a \mid q_aB$.
- Dann beginnt die Rückwärtsrechnung.
 - ▶ Wenn $\delta(q, a) = (q', b, \text{links})$, wähle für alle $c \in \Gamma$ die Produktion

$$q'cb \rightarrow cqa.$$

Ein Vorgänger der Konfiguration $* \dots * q'cb * \dots *$ ist $* \dots * cqa * \dots *$.

$G := (\Sigma, V, S, P)$ mit der Variablenmenge $V := (\Gamma \setminus \Sigma) \cup Q \cup \{\epsilon\}$ und $S := q_a$.

- Erzeuge den von M benutzten Bandbereich durch $q_a \rightarrow Bq_a \mid q_aB$.
- Dann beginnt die Rückwärtsrechnung.
 - ▶ Wenn $\delta(q, a) = (q', b, \text{links})$, wähle für alle $c \in \Gamma$ die Produktion

$$q'cb \rightarrow cqa.$$

Ein Vorgänger der Konfiguration $ \dots * q'cb * \dots *$ ist $* \dots * cqa * \dots *$.*

- ▶ Wenn $\delta(q, a) = (q', b, \text{bleib})$, fügen wir hinzu

$G := (\Sigma, V, S, P)$ mit der Variablenmenge $V := (\Gamma \setminus \Sigma) \cup Q \cup \{\epsilon\}$ und $S := q_a$.

- Erzeuge den von M benutzten Bandbereich durch $q_a \rightarrow Bq_a \mid q_aB$.
- Dann beginnt die Rückwärtsrechnung.
 - ▶ Wenn $\delta(q, a) = (q', b, \text{links})$, wähle für alle $c \in \Gamma$ die Produktion

$$q'cb \rightarrow cqa.$$

Ein Vorgänger der Konfiguration $ \dots * q'cb * \dots *$ ist $* \dots * cqa * \dots *$.*

- ▶ Wenn $\delta(q, a) = (q', b, \text{bleib})$, fügen wir hinzu

$$q'b \rightarrow qa$$

Ein Vorgänger der Konfiguration $ \dots * q'b * \dots *$ ist $* \dots * qa * \dots *$.*

$G := (\Sigma, V, S, P)$ mit der Variablenmenge $V := (\Gamma \setminus \Sigma) \cup Q \cup \{\epsilon\}$ und $S := q_a$.

- Erzeuge den von M benutzten Bandbereich durch $q_a \rightarrow Bq_a \mid q_aB$.
- Dann beginnt die Rückwärtsrechnung.
 - ▶ Wenn $\delta(q, a) = (q', b, \text{links})$, wähle für alle $c \in \Gamma$ die Produktion

$$q'cb \rightarrow cqa.$$

Ein Vorgänger der Konfiguration $ \dots * q'cb * \dots *$ ist $* \dots * cqa * \dots *$.*

- ▶ Wenn $\delta(q, a) = (q', b, \text{bleib})$, fügen wir hinzu

$$q'b \rightarrow qa$$

Ein Vorgänger der Konfiguration $ \dots * q'b * \dots *$ ist $* \dots * qa * \dots *$.*

- ▶ Wenn $\delta(q, a) = (q', b, \text{rechts})$, dann fügen wir hinzu

$G := (\Sigma, V, S, P)$ mit der Variablenmenge $V := (\Gamma \setminus \Sigma) \cup Q \cup \{\epsilon\}$ und $S := q_a$.

- Erzeuge den von M benutzten Bandbereich durch $q_a \rightarrow Bq_a \mid q_aB$.
- Dann beginnt die Rückwärtsrechnung.
 - ▶ Wenn $\delta(q, a) = (q', b, \text{links})$, wähle für alle $c \in \Gamma$ die Produktion

$$q'cb \rightarrow cqa.$$

Ein Vorgänger der Konfiguration $ \dots * q'cb * \dots *$ ist $* \dots * cqa * \dots *$.*

- ▶ Wenn $\delta(q, a) = (q', b, \text{bleib})$, fügen wir hinzu

$$q'b \rightarrow qa$$

Ein Vorgänger der Konfiguration $ \dots * q'b * \dots *$ ist $* \dots * qa * \dots *$.*

- ▶ Wenn $\delta(q, a) = (q', b, \text{rechts})$, dann fügen wir hinzu

$$bq' \rightarrow qa.$$

Ein Vorgänger der Konfiguration $ \dots * bq' * \dots *$ ist $* \dots * qa * \dots *$.*

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

- Die zusätzlichen Produktionen

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

- Die zusätzlichen Produktionen

$$q_0 \rightarrow L$$

$$B L \rightarrow L$$

$$L a \rightarrow aL \quad \text{für } a \in \Sigma$$

$$L B \rightarrow L$$

$$L \rightarrow \text{das leere Wort}$$

garantieren jetzt, dass das Wort w abgeleitet wird.

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

- Die zusätzlichen Produktionen

$$q_0 \rightarrow L$$

$$B L \rightarrow L$$

$$L a \rightarrow a L \quad \text{für } a \in \Sigma$$

$$L B \rightarrow L$$

$$L \rightarrow \text{das leere Wort}$$

garantieren jetzt, dass das Wort w abgeleitet wird.

- Wenn andererseits G eine Ableitung des Worts w besitzt, dann hat die Ableitung die Form

$$q_a \xrightarrow{*} B^k q_0 w B^s \xrightarrow{*} w$$

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

Kontextsensitive Grammatiken

Typ-1 Grammatiken

Alle Produktionen

$$u \rightarrow v$$

einer kontextsensitiven Grammatik sind **längenerhaltend**: Es gilt $|u| \leq |v|$.

Für jede Ableitung $S \xrightarrow{*} w$ sind alle zwischenzeitlich erzeugten Strings in ihrer Länge durch $|w|$ nach oben beschränkt.

Typ-1 Grammatiken

Alle Produktionen

$$u \rightarrow v$$

einer kontextsensitiven Grammatik sind **längenerhaltend**: Es gilt $|u| \leq |v|$.

Für jede Ableitung $S \xrightarrow{*} w$ sind alle zwischenzeitlich erzeugten Strings in ihrer Länge durch $|w|$ nach oben beschränkt.

- ▶ Eine Ableitungsfolge kann auf Platz $\mathcal{O}(|w|)$ geraten und verifiziert werden.
- ▶ Jede kontextsensitive Sprache kann durch eine nichtdeterministische Turingmaschine auf linearem Platz erkannt werden.

Typ-1 Grammatiken

Alle Produktionen

$$u \rightarrow v$$

einer kontextsensitiven Grammatik sind **längenerhaltend**: Es gilt $|u| \leq |v|$.

Für jede Ableitung $S \xrightarrow{*} w$ sind alle zwischenzeitlich erzeugten Strings in ihrer Länge durch $|w|$ nach oben beschränkt.

- ▶ Eine Ableitungsfolge kann auf Platz $\mathcal{O}(|w|)$ geraten und verifiziert werden.
- ▶ Jede kontextsensitive Sprache kann durch eine nichtdeterministische Turingmaschine auf linearem Platz erkannt werden.

Gibt es zu jeder Sprache $L \in \text{NSPACE}(n)$ (mit $\epsilon \notin L$) eine Typ-1 Grammatik G mit

$$L(G) = L?$$

Die Konstruktion der Typ-1 Grammatik

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

$$q_a \xrightarrow{*} B^k q_0 w B^s \iff w \in L(M). \quad (1)$$

gilt. Alle Produktionen für die Ableitung $q_a \xrightarrow{*} B^k q_0 w B^s$ sind längenerhaltend.

Die Konstruktion der Typ-1 Grammatik

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

$$q_a \xrightarrow{*} B^k q_0 w B^s \iff w \in L(M). \quad (1)$$

gilt. Alle Produktionen für die Ableitung $q_a \xrightarrow{*} B^k q_0 w B^s$ sind längenerhaltend.

- Wenn die nichtdeterministische TM M mit linearem Platz arbeitet, dann gibt es eine äquivalente „in-place“ TM M' :

Vergrößere das Bandalphabet entsprechend.

Die Konstruktion der Typ-1 Grammatik

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

$$q_a \xrightarrow{*} B^k q_0 w B^s \iff w \in L(M). \quad (1)$$

gilt. Alle Produktionen für die Ableitung $q_a \xrightarrow{*} B^k q_0 w B^s$ sind längenerhaltend.

- Wenn die nichtdeterministische 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)

$$q_a \xrightarrow{*} q_0 w \iff w \in L(M') = L(M)$$

Die Konstruktion der Typ-1 Grammatik

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

$$q_a \xrightarrow{*} B^k q_0 w B^s \iff w \in L(M). \quad (1)$$

gilt. Alle Produktionen für die Ableitung $q_a \xrightarrow{*} B^k q_0 w B^s$ sind längenerhaltend.

- Wenn die nichtdeterministische 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)

$$q_a \xrightarrow{*} q_0 w \iff w \in L(M') = L(M)$$

- ▶ Wenn $L(M) \in \text{NSPACE}(n)$, dann ist $q_0 L(M)$ kontextsensitiv.

Die Konstruktion der Typ-1 Grammatik

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

$$q_a \xrightarrow{*} B^k q_0 w B^s \iff w \in L(M). \quad (1)$$

gilt. Alle Produktionen für die Ableitung $q_a \xrightarrow{*} B^k q_0 w B^s$ sind längenerhaltend.

- Wenn die nichtdeterministische 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)

$$q_a \xrightarrow{*} q_0 w \iff w \in L(M') = L(M)$$

- ▶ Wenn $L(M) \in \text{NSPACE}(n)$, dann ist $q_0 L(M)$ kontextsensitiv.
- ▶ Zeige: Wenn $q_0 L(M)$ kontextsensitiv ist, dann auch $L(M) \setminus \{\epsilon\}$.

Typ-2 und Typ-3 Grammatiken

Typ-2 und Typ-3 Grammatiken

Eine Sprache L liegt genau dann in

LOGCFL ,

wenn $L \leq_{\text{LOG}} K$ für eine kontextfreie Sprache K gilt.

- (a) Es gibt eine NL -vollständige kontextfreie Sprache L .
- (b) $\text{NL} \subseteq \text{LOGCFL} \subseteq \text{DSPACE}(\log^2 n)$.

Typ-2 und Typ-3 Grammatiken

Eine Sprache L liegt genau dann in

LOGCFL ,

wenn $L \leq_{\text{LOG}} K$ für eine kontextfreie Sprache K gilt.

(a) Es gibt eine NL -vollständige kontextfreie Sprache L .

(b) $\text{NL} \subseteq \text{LOGCFL} \subseteq \text{DSPACE}(\log^2 n)$.

- ▶ Es gibt eine NL -vollständige kontextfreie Sprache $\implies \text{NL} \subseteq \text{LOGCFL}$ folgt.
- ▶ $\text{LOGCFL} \subseteq \text{DSPACE}(\log^2 n)$ wird mit anderen Methoden gezeigt.

Typ-2 und Typ-3 Grammatiken

Eine Sprache L liegt genau dann in

LOGCFL ,

wenn $L \leq_{\text{LOG}} K$ für eine kontextfreie Sprache K gilt.

(a) Es gibt eine NL -vollständige kontextfreie Sprache L .

(b) $\text{NL} \subseteq \text{LOGCFL} \subseteq \text{DSPACE}(\log^2 n)$.

- ▶ Es gibt eine NL -vollständige kontextfreie Sprache $\implies \text{NL} \subseteq \text{LOGCFL}$ folgt.
- ▶ $\text{LOGCFL} \subseteq \text{DSPACE}(\log^2 n)$ wird mit anderen Methoden gezeigt.

(c) Es gelte $s(n) = o(\log_2 \log_2 n)$. Dann folgt

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

Diagonalisierung

Eine Funktion

$$s : \mathbb{N} \rightarrow \mathbb{N}$$

heißt genau dann

platz-konstruierbar,

wenn

- $s(n) = \Omega(\log_2 n)$ und
- es eine deterministische I-O TM gibt, die für Eingabe 0^n genau $s(n)$ Zellen des Arbeitsbands markiert und $\leq s(n)$ Speicherplatz benötigt.

Eine Funktion

$$s : \mathbb{N} \rightarrow \mathbb{N}$$

heißt genau dann

platz-konstruierbar,

wenn

- $s(n) = \Omega(\log_2 n)$ und
- es eine deterministische I-O TM gibt, die für Eingabe 0^n genau $s(n)$ Zellen des Arbeitsbands markiert und $\leq s(n)$ Speicherplatz benötigt.

Wir zeigen, dass man mit mehr zur Verfügung stehendem Speicher mehr Entscheidungsprobleme lösen kann.

Die zentrale Idee: *Diagonalisierung*.

Die Funktion s sei platz-konstruierbar \implies

$DSPACE(o(s))$ ist eine echte Teilmenge von $DSPACE(s)$.

Die Funktion s sei platz-konstruierbar \implies

$DSPACE(o(s))$ ist eine echte Teilmenge von $DSPACE(s)$.

Die *Diagonalisierungsmethode*: M^* ist nicht auf Platz $o(s)$ simulierbar.

1. M^* bestimmt die Länge n der Eingabe w .
2. M^* steckt auf dem Arbeitsband $2s(n)$ Zellen ab.
/* Dies ist mit Speicherplatz $s(n)$ möglich, da s platz-konstruierbar ist. */
3. Wenn

$$w \neq \langle M \rangle 0^k$$

für eine I-O Turingmaschine M und eine Zahl $k \in \mathbb{N}$ ist, dann verwirft M^* .

/* $\langle M \rangle$ bezeichnet die „Gödelnummer“ der Turingmaschine M . */

Die Funktion s sei platz-konstruierbar \implies

$DSPACE(o(s))$ ist eine echte Teilmenge von $DSPACE(s)$.

Die *Diagonalisierungsmethode*: M^* ist nicht auf Platz $o(s)$ simulierbar.

1. M^* bestimmt die Länge n der Eingabe w .
2. M^* steckt auf dem Arbeitsband $2s(n)$ Zellen ab.
/* Dies ist mit Speicherplatz $s(n)$ möglich, da s platz-konstruierbar ist. */
3. Wenn
$$w \neq \langle M \rangle 0^k$$
für eine I-O Turingmaschine M und eine Zahl $k \in \mathbb{N}$ ist, dann verwirft M^* .
/* $\langle M \rangle$ bezeichnet die „Gödelnummer“ der Turingmaschine M . */
4. M^* simuliert M auf Eingabe $w = \langle M \rangle 0^k$ und beginnt die Simulation mit dem Kopf in der Mitte des abgesteckten Bereichs.
 - ▶ Wenn M irgendwann den abgesteckten Bereich verlässt oder mehr als $2^{s(n)}$ Schritte benötigt, dann

Die Funktion s sei platz-konstruierbar \implies

$DSPACE(o(s))$ ist eine echte Teilmenge von $DSPACE(s)$.

Die *Diagonalisierungsmethode*: M^* ist nicht auf Platz $o(s)$ simulierbar.

1. M^* bestimmt die Länge n der Eingabe w .
2. M^* steckt auf dem Arbeitsband $2s(n)$ Zellen ab.
/* Dies ist mit Speicherplatz $s(n)$ möglich, da s platz-konstruierbar ist. */

3. Wenn

$$w \neq \langle M \rangle 0^k$$

für eine I-O Turingmaschine M und eine Zahl $k \in \mathbb{N}$ ist, dann verwirft M^* .

/* $\langle M \rangle$ bezeichnet die „Gödelnummer“ der Turingmaschine M . */

4. M^* simuliert M auf Eingabe $w = \langle M \rangle 0^k$ und beginnt die Simulation mit dem Kopf in der Mitte des abgesteckten Bereichs.
 - ▶ Wenn M irgendwann den abgesteckten Bereich verlässt oder mehr als $2^{s(n)}$ Schritte benötigt, dann verwirft M^* . /* **Achtung**: Schleife! */
 - ▶ Ansonsten *akzeptiert* M^* die Eingabe w genau dann, wenn

Die Funktion s sei platz-konstruierbar \implies

$DSPACE(o(s))$ ist eine echte Teilmenge von $DSPACE(s)$.

Die *Diagonalisierungsmethode*: M^* ist nicht auf Platz $o(s)$ simulierbar.

1. M^* bestimmt die Länge n der Eingabe w .
2. M^* steckt auf dem Arbeitsband $2s(n)$ Zellen ab.
/* Dies ist mit Speicherplatz $s(n)$ möglich, da s platz-konstruierbar ist. */

3. Wenn

$$w \neq \langle M \rangle 0^k$$

für eine I-O Turingmaschine M und eine Zahl $k \in \mathbb{N}$ ist, dann verwirft M^* .

/* $\langle M \rangle$ bezeichnet die „Gödelnummer“ der Turingmaschine M . */

4. M^* simuliert M auf Eingabe $w = \langle M \rangle 0^k$ und beginnt die Simulation mit dem Kopf in der Mitte des abgesteckten Bereichs.
 - ▶ Wenn M irgendwann den abgesteckten Bereich verlässt oder mehr als $2^{s(n)}$ Schritte benötigt, dann verwirft M^* . /* **Achtung**: Schleife! */
 - ▶ Ansonsten *akzeptiert* M^* die Eingabe w genau dann, wenn M **verwirft**.

Die Funktion s sei platz-konstruierbar \implies

$DSPACE(o(s))$ ist eine echte Teilmenge von $DSPACE(s)$.

Die *Diagonalisierungsmethode*: M^* ist nicht auf Platz $o(s)$ simulierbar.

1. M^* bestimmt die Länge n der Eingabe w .
2. M^* steckt auf dem Arbeitsband $2s(n)$ Zellen ab.
/* Dies ist mit Speicherplatz $s(n)$ möglich, da s platz-konstruierbar ist. */

3. Wenn

$$w \neq \langle M \rangle 0^k$$

für eine I-O Turingmaschine M und eine Zahl $k \in \mathbb{N}$ ist, dann verwirft M^* .

/* $\langle M \rangle$ bezeichnet die „Gödelnummer“ der Turingmaschine M . */

4. M^* simuliert M auf Eingabe $w = \langle M \rangle 0^k$ und beginnt die Simulation mit dem Kopf in der Mitte des abgesteckten Bereichs.
 - ▶ Wenn M irgendwann den abgesteckten Bereich verlässt oder mehr als $2^{s(n)}$ Schritte benötigt, dann verwirft M^* . /* **Achtung**: Schleife! */
 - ▶ Ansonsten *akzeptiert* M^* die Eingabe w genau dann, wenn M **verwirft**.

/* Auf Platz $O(s(n))$: Zähle bis $2^{s(n)} - 1$ und simuliere M . */

- (a) M^* hält immer und kommt mit Speicherplatzbedarf $\mathcal{O}(s(n))$ aus. $\implies L(M^*) \in \text{DSPACE}(s)$.

- (a) M^* hält immer und kommt mit Speicherplatzbedarf $\mathcal{O}(s(n))$ aus. $\implies L(M^*) \in \text{DSPACE}(s)$.
- (b) Und wenn $L(M^*) = L(M)$ für eine I-O TM M mit Speicherplatz $o(s)$?

- (a) M^* hält immer und kommt mit Speicherplatzbedarf $\mathcal{O}(s(n))$ aus. $\implies L(M^*) \in \text{DSPACE}(s)$.
- (b) Und wenn $L(M^*) = L(M)$ für eine I-O TM M mit Speicherplatz $o(s)$?
- ▶ Für *genügend große Eingabelänge* n rechnet M stets in Zeit $\leq 2^s \implies M^*$ kann M für Eingaben $w = \langle M \rangle 0^k$ mit *genügend großem* k simulieren.

- (a) M^* hält immer und kommt mit Speicherplatzbedarf $\mathcal{O}(s(n))$ aus. $\implies L(M^*) \in \text{DSPACE}(s)$.
- (b) Und wenn $L(M^*) = L(M)$ für eine I-O TM M mit Speicherplatz $o(s)$?
- ▶ Für *genügend große Eingabelänge* n rechnet M stets in Zeit $\leq 2^s \implies M^*$ kann M für Eingaben $w = \langle M \rangle 0^k$ mit *genügend großem* k simulieren.
 - ▶ Schritt (4) garantiert, dass sich $L(M)$ und $L(M^*)$ unterscheiden: Speicherplatz $o(s)$ ist unzureichend für die Berechnung von $L(M^*)$. □

Der Satz von Savitch (1970)

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

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

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

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

- 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) stellt sich als direkte Konsequenz von Teil (a) heraus: D-REACHABILITY nimmt eine Schlüsselrolle ein.

Der **Algorithmus von Savitch** überprüft, ob der Eingabegraph G einen Weg von *Knoten* u nach *Knoten* v der *genauen Länge* m besitzt.

Der **Algorithmus von Savitch** überprüft, ob der Eingabegraph G einen Weg von *Knoten* u nach *Knoten* v der *genauen Länge* m besitzt.

Savitch (u, v, m) :

Der **Algorithmus von Savitch** überprüft, ob der Eingabegraph G einen Weg von *Knoten* u nach *Knoten* v der *genauen Länge* m besitzt.

Savitch (u, v, m) :

Wenn $m = 1$, dann akzeptiere, wenn (u, v) eine Kante ist.

Der **Algorithmus von Savitch** überprüft, ob der Eingabegraph G einen Weg von *Knoten* u nach *Knoten* v der *genauen Länge* m besitzt.

Savitch (u, v, m) :

Wenn $m = 1$, dann akzeptiere, wenn (u, v) eine Kante ist.
Sonst akzeptiere wenn es einen Knoten w gibt, so dass Savitch($u, w, \lceil \frac{m}{2} \rceil$) und Savitch($w, v, \lfloor \frac{m}{2} \rfloor$) akzeptieren.

Der **Algorithmus von Savitch** überprüft, ob der Eingabegraph G einen Weg von *Knoten* u nach *Knoten* v der *genauen Länge* m besitzt.

Savitch (u, v, m) :

Wenn $m = 1$, dann akzeptiere, wenn (u, v) eine Kante ist.
Sonst akzeptiere wenn es einen Knoten w gibt, so dass Savitch($u, w, \lceil \frac{m}{2} \rceil$) und Savitch($w, v, \lfloor \frac{m}{2} \rfloor$) akzeptieren.

- ✓ Bei n Knoten gibt es bis zu $n - 1$ Masteraufrufe, jeweils für $u = 1, v = 2$ und für alle Werte von m mit $1 \leq m \leq n - 1$.
- ? Wie viel Speicherplatz wird benötigt?

Analyse von Speicherplatz und Laufzeit:

- Speicherplatzverbrauch:

Analyse von Speicherplatz und Laufzeit:

- Speicherplatzverbrauch:
 - ▶ Ein Stack der maximalen Höhe $\lceil \log_2 n \rceil$ genügt.

Analyse von Speicherplatz und Laufzeit:

- Speicherplatzverbrauch:
 - ▶ Ein Stack der maximalen Höhe $\lceil \log_2 n \rceil$ genügt.
 - ▶ Jedes Element des Stacks entspricht zwei Knoten und einer Zahl $m \leq n$:
Logarithmischer Speicher reicht für Stackelemente.
 - ▶ Insgesamt benötigen wir Speicherplatz $\mathcal{O}(\log_2^2 n)$.

Analyse von Speicherplatz und Laufzeit:

- Speicherplatzverbrauch:
 - ▶ Ein Stack der maximalen Höhe $\lceil \log_2 n \rceil$ genügt.
 - ▶ Jedes Element des Stacks entspricht zwei Knoten und einer Zahl $m \leq n$:
Logarithmischer Speicher reicht für Stackelemente.
 - ▶ Insgesamt benötigen wir Speicherplatz $\mathcal{O}(\log_2^2 n)$.
- Die Laufzeit ist höchstens exponentiell im Speicherplatz und deshalb durch $2^{\mathcal{O}(\log_2^2 n)}$ beschränkt. Furchtbar, aber was soll's?

Analyse von Speicherplatz und Laufzeit:

- Speicherplatzverbrauch:
 - ▶ Ein Stack der maximalen Höhe $\lceil \log_2 n \rceil$ genügt.
 - ▶ Jedes Element des Stacks entspricht zwei Knoten und einer Zahl $m \leq n$:
Logarithmischer Speicher reicht für Stackelemente.
 - ▶ Insgesamt benötigen wir Speicherplatz $\mathcal{O}(\log_2^2 n)$.
- Die Laufzeit ist höchstens exponentiell im Speicherplatz und deshalb durch $2^{\mathcal{O}(\log_2^2 n)}$ beschränkt. Furchtbar, aber was soll's?

Warum gilt $\text{NSPACE}(s) \subseteq \text{DSPACE}(s^2)$, wenn s platzkonstrierbar ist?

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

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

- M sei eine beliebige nichtdeterministische Turingmaschine mit Speicherplatzbedarf s und w sei eine Eingabe.
- Wir konstruieren eine deterministische Turingmaschine M^* , die M auf Eingabe w mit Speicherplatz $\mathcal{O}(s^2)$ simuliert.

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

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

- M sei eine beliebige nichtdeterministische Turingmaschine mit Speicherplatzbedarf s und w sei eine Eingabe.
- Wir konstruieren eine deterministische Turingmaschine M^* , die M auf Eingabe w mit Speicherplatz $\mathcal{O}(s^2)$ simuliert. Da s platz-konstruierbar ist,
 - ▶ kann M^* einen Speicherplatz von $s(n)$ Zellen abstecken und jeden 1-Schritt-Übergang zwischen zwei Konfigurationen von M überprüfen.

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

- M sei eine beliebige nichtdeterministische Turingmaschine mit Speicherplatzbedarf s und w sei eine Eingabe.
- Wir konstruieren eine deterministische Turingmaschine M^* , die M auf Eingabe w mit Speicherplatz $\mathcal{O}(s^2)$ simuliert. Da s platz-konstruierbar ist,
 - ▶ kann M^* einen Speicherplatz von $s(n)$ Zellen abstecken und jeden 1-Schritt-Übergang zwischen zwei Konfigurationen von M überprüfen.
 - ▶ M^* führt den Algorithmus von Savitch aus und zwar
 - für den Berechnungsgraphen $G_M(w)$ mit Knoten 1,2 entsprechend gewählt.
 - ▶ M^* akzeptiert w : \iff es gibt einen Weg in $G_M(w)$ von Knoten 1 nach 2.

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

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

- M sei eine beliebige nichtdeterministische Turingmaschine mit Speicherplatzbedarf s und w sei eine Eingabe.
- Wir konstruieren eine deterministische Turingmaschine M^* , die M auf Eingabe w mit Speicherplatz $\mathcal{O}(s^2)$ simuliert. Da s platz-konstruierbar ist,
 - ▶ kann M^* einen Speicherplatz von $s(n)$ Zellen abstecken und jeden 1-Schritt-Übergang zwischen zwei Konfigurationen von M überprüfen.
 - ▶ M^* führt den Algorithmus von Savitch aus und zwar
 - für den Berechnungsgraphen $G_M(w)$ mit Knoten 1,2 entsprechend gewählt.
 - ▶ M^* akzeptiert w : \iff es gibt einen Weg in $G_M(w)$ von Knoten 1 nach 2.

Für Graphen mit $K = 2^{\mathcal{O}(s(n))}$ Knoten genügt Speicherplatz $\mathcal{O}(\log_2^2 K) = \mathcal{O}(s(n)^2)$.

Der Satz von Immerman und Szelepcsényi (1988)

(a) D-UNREACHABILITY

Akzeptiere einen gerichteten Graphen G genau dann, wenn G **keinen** Weg von Knoten 1 nach Knoten 2 besitzt

gehört zur Klasse NL .

(b) Die Funktion s sei platz-konstruierbar. Dann gilt für alle Sprachen L ,

$$L \in NSPACE(s) \Leftrightarrow \bar{L} \in NSPACE(s)$$

Nichtdeterministischer Speicherplatz ist abgeschlossen unter Komplementbildung.

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. Zeige: D-UNREACHABILITY liegt in NL .
- 2 Im zweiten Schritt lösen wir das *Anzahlproblem* in NL .

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. Zeige: D-UNREACHABILITY liegt in NL .
- 2 Im zweiten Schritt lösen wir das *Anzahlproblem* in NL .
 - m Knoten seien vom Knoten 1 aus erreichbar.
 - Die Idee und ihre Implementierung.
 - ▶ Rate *nacheinander genau* m von Knoten 1 aus erreichbare Knoten. Wie?

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. Zeige: D-UNREACHABILITY liegt in NL .
 - 2 Im zweiten Schritt lösen wir das *Anzahlproblem* in NL .
- m Knoten seien vom Knoten 1 aus erreichbar.
 - Die Idee und ihre Implementierung.
 - ▶ Rate *nacheinander genau* m von Knoten 1 aus erreichbare Knoten. Wie?
 - ★ Für v_i rate einen Weg $1 \xrightarrow{*} v_i$.
 - ★ Wenn dies erfolgreich war, rate danach v_{i+1} und wiederhole das Vorgehen für v_{i+1} .

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. Zeige: D-UNREACHABILITY liegt in NL .
 - 2 Im zweiten Schritt lösen wir das *Anzahlproblem* in NL .
- m Knoten seien vom Knoten 1 aus erreichbar.
 - Die Idee und ihre Implementierung.
 - ▶ Rate *nacheinander genau* m von Knoten 1 aus erreichbare Knoten. Wie?
 - ★ Für v_i rate einen Weg $1 \xrightarrow{*} v_i$.
 - ★ Wenn dies erfolgreich war, rate danach v_{i+1} und wiederhole das Vorgehen für v_{i+1} . Achtung: Fordere $v_i < v_{i+1}$.

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. Zeige: D-UNREACHABILITY liegt in NL .
- 2 Im zweiten Schritt lösen wir das *Anzahlproblem* in NL .

- m Knoten seien vom Knoten 1 aus erreichbar.
- Die Idee und ihre Implementierung.
 - ▶ Rate *nacheinander genau* m von Knoten 1 aus erreichbare Knoten. Wie?
 - ★ Für v_i rate einen Weg $1 \xrightarrow{*} v_i$.
 - ★ Wenn dies erfolgreich war, rate danach v_{i+1} und wiederhole das Vorgehen für v_{i+1} . Achtung: Fordere $v_i < v_{i+1}$.
 - ▶ Wenn Knoten 2 von v_1, \dots, v_m verschieden ist, dann ist Knoten 2 **nicht** von Knoten 1 aus erreichbar: Wir akzeptieren.

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

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 $m_{n-1} = ?$ ist zu bestimmen.
- Zeige, dass m_{i+1} in NL berechnet werden kann, wenn m_i bekannt ist.

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 $m_{n-1} = ?$ ist zu bestimmen.
- Zeige, dass m_{i+1} in NL berechnet werden kann, wenn m_i bekannt ist.
 - ▶ Setze zu Anfang $m_{i+1} = 0$.

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 $m_{n-1} = ?$ ist zu bestimmen.
- Zeige, dass m_{i+1} in NL berechnet werden kann, wenn m_i bekannt ist.
 - ▶ Setze zu Anfang $m_{i+1} = 0$.
 - ▶ Verarbeite die Knoten k in *aufsteigender* Reihenfolge.

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 $m_{n-1} = ?$ ist zu bestimmen.
- Zeige, dass m_{i+1} in NL berechnet werden kann, wenn m_i bekannt ist.
 - ▶ Setze zu Anfang $m_{i+1} = 0$.
 - ▶ Verarbeite die Knoten k in *aufsteigender* Reihenfolge.
 - ★ Rate nacheinander die m_i von 1 durch einen Weg der Länge höchstens m_i erreichbaren Knoten $v_{i,r}$ mit $v_{i,1} < \dots < v_{i,m_i}$.
 - ★ Verifiziere, dass $v_{i,r}$ durch einen Weg der Länge höchstens m_i erreichbar ist.
 - ★ Überprüfe, ob

$(v_{i,r}, k)$ eine Kante ist oder $v_{i,r} = k$ gilt.

Wenn ja, setze $m_{i+1} :=$

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 $m_{n-1} = ?$ ist zu bestimmen.
- Zeige, dass m_{i+1} in NL berechnet werden kann, wenn m_i bekannt ist.
 - ▶ Setze zu Anfang $m_{i+1} = 0$.
 - ▶ Verarbeite die Knoten k in *aufsteigender* Reihenfolge.
 - ★ Rate nacheinander die m_i von 1 durch einen Weg der Länge höchstens m_i erreichbaren Knoten $v_{i,r}$ mit $v_{i,1} < \dots < v_{i,m_i}$.
 - ★ Verifiziere, dass $v_{i,r}$ durch einen Weg der Länge höchstens m_i erreichbar ist.
 - ★ Überprüfe, ob

$(v_{i,r}, k)$ eine Kante ist oder $v_{i,r} = k$ gilt.

Wenn ja, setze $m_{i+1} := m_{i+1} + 1$ und $k := k + 1$. Ansonsten

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 $m_{n-1} = ?$ ist zu bestimmen.
- Zeige, dass m_{i+1} in NL berechnet werden kann, wenn m_i bekannt ist.
 - ▶ Setze zu Anfang $m_{i+1} = 0$.
 - ▶ Verarbeite die Knoten k in *aufsteigender* Reihenfolge.
 - ★ Rate nacheinander die m_i von 1 durch einen Weg der Länge höchstens m_i erreichbaren Knoten $v_{i,r}$ mit $v_{i,1} < \dots < v_{i,m_i}$.
 - ★ Verifiziere, dass $v_{i,r}$ durch einen Weg der Länge höchstens m_i erreichbar ist.
 - ★ Überprüfe, ob

$(v_{i,r}, k)$ eine Kante ist oder $v_{i,r} = k$ gilt.

Wenn ja, setze $m_{i+1} := m_{i+1} + 1$ und $k := k + 1$. Ansonsten rate $v_{i,r+1}$ und fahre mit der Untersuchung von k fort.

Die Funktion $s : \mathbb{N} \rightarrow \mathbb{N}$ sei platz-konstruierbar.

- L werde von einer nichtdeterministischen Turingmaschine M mit Speicherplatzbedarf $s \geq \log_2 n$ erkannt.
- Dann bauen wir eine nichtdeterministische TM M^* , die das Komplement \bar{L} mit Speicherplatzbedarf $\mathcal{O}(s)$ erkennt.

Die Funktion $s : \mathbb{N} \rightarrow \mathbb{N}$ sei platz-konstruierbar.

- L werde von einer nichtdeterministischen Turingmaschine M mit Speicherplatzbedarf $s \geq \log_2 n$ erkannt.
 - Dann bauen wir eine nichtdeterministische TM M^* , die das Komplement \bar{L} mit Speicherplatzbedarf $\mathcal{O}(s)$ erkennt.
-
- M^* muss genau dann akzeptieren, wenn es keine akzeptierende Berechnung gibt, wenn also $G_M(w)$ zu D-UNREACHABILITY gehört.

Die Funktion $s : \mathbb{N} \rightarrow \mathbb{N}$ sei platz-konstruierbar.

- L werde von einer nichtdeterministischen Turingmaschine M mit Speicherplatzbedarf $s \geq \log_2 n$ erkannt.
 - Dann bauen wir eine nichtdeterministische TM M^* , die das Komplement \bar{L} mit Speicherplatzbedarf $\mathcal{O}(s)$ erkennt.
- M^* muss genau dann akzeptieren, wenn es keine akzeptierende Berechnung gibt, wenn also $G_M(w)$ zu D-UNREACHABILITY gehört.
 - ▶ M^* wendet den Algorithmus von Immerman und Szelepcsényi für den Berechnungsgraphen $G_M(w)$ an.
 - ▶ M^* muss klären, welche Kanten in $G_M(w)$ einzusetzen sind:

Die Funktion $s : \mathbb{N} \rightarrow \mathbb{N}$ sei platz-konstruierbar.

- L werde von einer nichtdeterministischen Turingmaschine M mit Speicherplatzbedarf $s \geq \log_2 n$ erkannt.
 - Dann bauen wir eine nichtdeterministische TM M^* , die das Komplement \bar{L} mit Speicherplatzbedarf $\mathcal{O}(s)$ erkennt.
- M^* muss genau dann akzeptieren, wenn es keine akzeptierende Berechnung gibt, wenn also $G_M(w)$ zu D-UNREACHABILITY gehört.
 - ▶ M^* wendet den Algorithmus von Immerman und Szelepcsényi für den Berechnungsgraphen $G_M(w)$ an.
 - ▶ M^* muss klären, welche Kanten in $G_M(w)$ einzusetzen sind: Platz $\mathcal{O}(s)$ reicht, da die Konfigurationen von M nur Platz $\mathcal{O}(s)$ benötigen.

Die Funktion $s : \mathbb{N} \rightarrow \mathbb{N}$ sei platz-konstruierbar.

- L werde von einer nichtdeterministischen Turingmaschine M mit Speicherplatzbedarf $s \geq \log_2 n$ erkannt.
 - Dann bauen wir eine nichtdeterministische TM M^* , die das Komplement \bar{L} mit Speicherplatzbedarf $\mathcal{O}(s)$ erkennt.
-
- M^* muss genau dann akzeptieren, wenn es keine akzeptierende Berechnung gibt, wenn also $G_M(w)$ zu D-UNREACHABILITY gehört.
 - ▶ M^* wendet den Algorithmus von Immerman und Szelepcsényi für den Berechnungsgraphen $G_M(w)$ an.
 - ▶ M^* muss klären, welche Kanten in $G_M(w)$ einzusetzen sind: Platz $\mathcal{O}(s)$ reicht, da die Konfigurationen von M nur Platz $\mathcal{O}(s)$ benötigen.
 - Also genügt insgesamt Platz $\mathcal{O}(s)$.

PSPACE und PSPACE-Vollständigkeit

Warum PSPACE? In PSPACE kann man

- (einige) Spiele optimal spielen,
- das Äquivalenzproblem für NFAs und reguläre Ausdrücke lösen,
- Zählprobleme lösen: „Wieviele perfekte Matchings hat ein ungerichteter Graph?“
- effiziente probabilistische Rechner oder Quantenrechner simulieren.

Gilt $P = PSPACE$?

Wie sehen die schwierigsten Probleme in dieser Riesenklasse aus?

$P \stackrel{?}{=} PSPACE$: PSPACE-Vollständigkeit

Die schwierigsten Sprachen in PSPACE für die *polynomielle Reduktion*:

- (a) Eine Sprache L heißt **PSPACE-hart**, falls $K \leq_P L$ für alle Sprachen $K \in PSPACE$ gilt.
- (b) L heißt **PSPACE-vollständig**, falls L PSPACE-hart ist und falls $L \in PSPACE$.

$P \stackrel{?}{=} PSPACE$: PSPACE-Vollständigkeit

Die schwierigsten Sprachen in PSPACE für die *polynomielle Reduktion*:

- (a) Eine Sprache L heißt **PSPACE-hart**, falls $K \leq_P L$ 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 $L \leq_P K$ und wenn $K \in P$, dann ist

$P \stackrel{?}{=} PSPACE$: PSPACE-Vollständigkeit

Die schwierigsten Sprachen in PSPACE für die *polynomielle Reduktion*:

- (a) Eine Sprache L heißt **PSPACE-hart**, falls $K \leq_P L$ 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 $L \leq_P K$ und wenn $K \in P$, dann ist auch $L \in P$.

$P \stackrel{?}{=} PSPACE$: PSPACE-Vollständigkeit

Die schwierigsten Sprachen in PSPACE für die *polynomielle Reduktion*:

- (a) Eine Sprache L heißt **PSPACE-hart**, falls $K \leq_P L$ 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 $L \leq_P K$ 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

$$L \in P \iff P = PSPACE.$$

- Wenn $M \leq_P K$ und $K \leq_P L$, dann ist

$P \stackrel{?}{=} PSPACE$: PSPACE-Vollständigkeit

Die schwierigsten Sprachen in PSPACE für die *polynomielle Reduktion*:

- (a) Eine Sprache L heißt **PSPACE-hart**, falls $K \leq_P L$ 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 $L \leq_P K$ 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

$$L \in P \iff P = PSPACE.$$

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

QBF: Quantifizierte Boolesche Formeln

Quantifizierte Boolesche Formeln

Eine **quantifizierte Boolesche Formel** ϕ besteht aus

- einer Folge von Existenz- und Allquantoren,
- gefolgt von einer *aussagenlogischen Formel* α . Jede in α vorkommende Variable wird von genau einem Quantor gebunden.

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

Quantifizierte Boolesche Formeln

Eine **quantifizierte Boolesche Formel** ϕ besteht aus

- einer Folge von 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

Quantifizierte Boolesche Formeln

Eine **quantifizierte Boolesche Formel** ϕ besteht aus

- einer Folge von 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.
 - ▶ Also ist $\phi \notin \text{QBF}$.

Quantifizierte Boolesche Formeln

Eine **quantifizierte Boolesche Formel** ϕ besteht aus

- einer Folge von 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.
 - ▶ Also ist $\phi \notin \text{QBF}$.
- Die Formel $\psi \equiv \forall p \exists q ((p \vee \neg q) \wedge (\neg p \vee q))$ ist

Quantifizierte Boolesche Formeln

Eine **quantifizierte Boolesche Formel** ϕ besteht aus

- einer Folge von 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.
 - ▶ Also ist $\phi \notin \text{QBF}$.
- 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 Wert für q .
 - ▶ Also ist $\psi \in \text{QBF}$.

Alternierende Polynomialzeitberechnungen

Zur Erinnerung: AP ist die Klasse aller in polynomieller Zeit durch alternierende Algorithmen entscheidbaren Probleme. Es gilt

$$QBF \in AP = PSPACE$$

Alternierende Polynomialzeitberechnungen

Zur Erinnerung: AP ist die Klasse aller in polynomieller Zeit durch alternierende Algorithmen entscheidbaren Probleme. Es gilt

$$QBF \in AP = PSPACE$$

- $AP \subseteq PSPACE$: Werte den Berechnungsbaum eines alternierenden Algorithmus mit Laufzeit t durch einen deterministischen Algorithmus mit Speicherplatz-Komplexität $\mathcal{O}(t)$ aus.
- $PSPACE \subseteq AP$:
 - ▶ Zeige $QBF \in AP$ und
 - ▶ dass QBF ein $PSPACE$ -vollständiges Problem ist.

Alternierende Polynomialzeitberechnungen

Zur Erinnerung: AP ist die Klasse aller in polynomieller Zeit durch alternierende Algorithmen entscheidbaren Probleme. Es gilt

$$QBF \in AP = PSPACE$$

- $AP \subseteq PSPACE$: Werte den Berechnungsbaum eines alternierenden Algorithmus mit Laufzeit t durch einen deterministischen Algorithmus mit Speicherplatz-Komplexität $\mathcal{O}(t)$ aus.
- $PSPACE \subseteq AP$:
 - ▶ Zeige $QBF \in AP$ und
 - ▶ dass QBF ein $PSPACE$ -vollständiges Problem ist.

Die Sichtweise $AP = PSPACE$ motiviert die Wahl von QBF .

QBF gehört zu PSPACE.

Zeige: *QBF ist PSPACE-hart*:

- Die Sprache $L = L(M) \in \text{PSPACE}$ werde von einer deterministischen Turingmaschine M mit Speicherplatzbedarf $\mathcal{O}(n^k)$ berechnet.

QBF gehört zu PSPACE.

Zeige: *QBF ist PSPACE-hart*:

- Die Sprache $L = L(M) \in \text{PSPACE}$ werde von einer deterministischen Turingmaschine M mit Speicherplatzbedarf $\mathcal{O}(n^k)$ berechnet.
- Zeige

$$L \leq_P \text{QBF} :$$

Baue für jede Eingabe w von L in polynomieller Zeit eine quantifizierte Boolesche Formel ϕ_w mit

QBF gehört zu PSPACE.

Zeige: *QBF ist PSPACE-hart*:

- Die Sprache $L = L(M) \in \text{PSPACE}$ werde von einer deterministischen Turingmaschine M mit Speicherplatzbedarf $\mathcal{O}(n^k)$ berechnet.
- Zeige

$$L \leq_P \text{QBF} :$$

Baue für jede Eingabe w von L in polynomieller Zeit eine quantifizierte Boolesche Formel ϕ_w mit

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

Wir müssen eine Formel ϕ_w bauen mit

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

Wir müssen eine Formel ϕ_w bauen mit

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

- Wir wissen: Es ist $L = L(M)$ für eine deterministische TM M mit Speicherplatzbedarf $s = \mathcal{O}(n^k)$.

Wir müssen eine Formel ϕ_w bauen mit

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

- Wir wissen: Es ist $L = L(M)$ für eine deterministische TM M mit Speicherplatzbedarf $s = \mathcal{O}(n^k)$.
- Wie im NP-Vollständigkeitsbeweis von KNF-SAT führen wir aussagenlogische Variablen ein wie
 - $Kopf^t(z)$ für die Kopfposition. $Kopf^t(z)$ soll genau dann wahr sein, 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 sein, 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 sein, wenn q der Zustand zum Zeitpunkt ist.
- Aber M kann bis zu $2^{\mathcal{O}(n^k)}$ Schritte ausführen!!!

- (a) Für Konfigurationen c und d schreibe

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

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

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

- (a) Für Konfigurationen c und d schreibe

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

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

- (b) Baue eine höchstens polynomiell (polynomiell in „was“?) 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 gilt

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

- Setze

$$\phi_w :=$$

- (a) Für Konfigurationen c und d schreibe

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

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

- (b) Baue eine höchstens polynomiell (polynomiell in „was“?) 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 gilt

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

- Setze

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

Und jetzt der Clou:

$$\psi_{2^{t+1}}(c, d) \equiv$$

Und jetzt der Clou:

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

Und jetzt der Clou:

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

Und jetzt der Clou:

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

- $\exists e$ entspricht der *Folge* der Existenz-Quantoren zu den Variablen
 - ▶ Kopfposition, Zelleninhalt und Zustand der Konfiguration e .Ähnliches gilt für $\forall f$ und $\forall g$.

Und jetzt der Clou:

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

- $\exists e$ 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 .

Und jetzt der Clou:

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

- $\exists e$ 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 $\mathcal{O}(n^k)$, also höchstens um den Speicherplatzbedarf von M .

Und jetzt der Clou:

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

- $\exists e$ 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 $\mathcal{O}(n^k)$, also höchstens um den Speicherplatzbedarf von M .

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

Das GEOGRAPHIE-Spiel

In „Geographie“ wählen Alice und Bob abwechselnd bisher nicht genannte Städtenamen: Jede Stadt muss mit dem letzten Buchstaben der zuvor genannten Stadt beginnen.

In „**Geographie**“ wählen Alice und Bob abwechselnd bisher nicht genannte Städtenamen: Jede Stadt muss mit dem letzten Buchstaben der zuvor genannten Stadt beginnen.

- Die **Eingabe**: Ein gerichteter Graph $G = (V, E)$ und ein Knoten $s \in V$.
- Die **Spielregeln**:
 - ▶ Alice und Bob wählen abwechselnd eine noch nicht benutzte Kante aus E .
 - ★ Alice beginnt und wählt eine Kante mit Startknoten s .
 - ★ Jede anschließend gewählte Kante muß im Endknoten der zuvor gewählten Kante beginnen.
 - ★ Kein Knoten darf zweimal besucht werden.
 - ▶ Der Spieler verliert, der als erster nicht mehr ziehen kann.

Das GEOGRAPHIE-Spiel

In „Geographie“ wählen Alice und Bob abwechselnd bisher nicht genannte Städtenamen: Jede Stadt muss mit dem letzten Buchstaben der zuvor genannten Stadt beginnen.

- Die **Eingabe**: Ein gerichteter Graph $G = (V, E)$ und ein Knoten $s \in V$.
- Die **Spielregeln**:
 - ▶ Alice und Bob wählen abwechselnd eine noch nicht benutzte Kante aus E .
 - ★ Alice beginnt und wählt eine Kante mit Startknoten s .
 - ★ Jede anschließend gewählte Kante muß im Endknoten der zuvor gewählten Kante beginnen.
 - ★ Kein Knoten darf zweimal besucht werden.
 - ▶ Der Spieler verliert, der als erster nicht mehr ziehen kann.

Geographie ist $PSPACE$ -vollständig.

- Wir haben schon beobachtet, dass sich QBF als ein Spiel zwischen dem Existenzquantor und dem Allquantor auffassen lässt.

- 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 für ein interessantes, weil sehr schwieriges Spiel.

Leider kann die Komplexitätstheorie keine Aussagen über Spiele mit fixierter Spielgröße machen.

- 1 Für eine nichtdeterministische Turingmaschine M und eine Eingabe $x \in \Sigma^*$ ist $a_M(x)$ die Anzahl akzeptierender Berechnungen von M auf Eingabe x .
- 2 Das Zählproblem zu M ist die Menge

$$Z(M) := \{ (x, a_M(x)) : x \in \Sigma^* \}.$$

- 3 $\#P$ (gesprochen Sharp P) ist die Klasse aller Zählprobleme $Z(M)$ für einen in Polynomialzeit rechnenden nichtdeterministischen Algorithmus M .

- 1 Für eine nichtdeterministische Turingmaschine M und eine Eingabe $x \in \Sigma^*$ ist $a_M(x)$ die Anzahl akzeptierender Berechnungen von M auf Eingabe x .
- 2 Das Zählproblem zu M ist die Menge

$$Z(M) := \{ (x, a_M(x)) : x \in \Sigma^* \}.$$

- 3 $\#P$ (gesprochen Sharp P) ist die Klasse aller Zählprobleme $Z(M)$ für einen in Polynomialzeit rechnenden nichtdeterministischen Algorithmus M .
- Wenn M nichtdeterministisch in polynomieller Zeit rechnet, dann kann die Funktion a_M auf polynomiellem Speicherplatz berechnet werden.

- 1 Für eine nichtdeterministische Turingmaschine M und eine Eingabe $x \in \Sigma^*$ ist $a_M(x)$ die Anzahl akzeptierender Berechnungen von M auf Eingabe x .
- 2 Das Zählproblem zu M ist die Menge

$$Z(M) := \{ (x, a_M(x)) : x \in \Sigma^* \}.$$

- 3 $\#P$ (gesprochen Sharp P) ist die Klasse aller Zählprobleme $Z(M)$ für einen in Polynomialzeit rechnenden nichtdeterministischen Algorithmus M .
- Wenn M nichtdeterministisch in polynomieller Zeit rechnet, dann kann die Funktion a_M auf polynomiellem Speicherplatz berechnet werden.
 - Für nichtdeterm. Turingmaschinen M_1, M_2 sagt man, dass $Z(M_1)$ auf $Z(M_2)$ reduzierbar ist, wenn es effiziente, determ. Transformationen T_1, T_2 gibt, so dass

$$a_{M_1}(x) = T_2(a_{M_2}(T_1(x)))$$

und schreibt $Z(M_1) \leq_Z Z(M_2)$.

Sharp-P vollständige Probleme

Ein Zählproblem $Z(M)$ ist vollständig für Sharp-P genau dann, wenn

$$Z(K) \leq_z Z(M)$$

für alle Zählprobleme $Z(K)$ in Sharp-P gilt.

Sharp-P vollständige Probleme

Ein Zählproblem $Z(M)$ ist vollständig für Sharp-P genau dann, wenn

$$Z(K) \leq_Z Z(M)$$

für alle Zählprobleme $Z(K)$ in Sharp-P gilt.

Die folgenden Zählprobleme sind Sharp-P vollständig:

- die Permanente einer $n \times n$ -Matrix A von Nullen und Einsen, also der Wert

$$\sum_{\pi \in S_n} \prod_{i=1}^n A_{i, \pi(i)}.$$

S_n ist die Menge aller Permutationen der Menge $\{1, \dots, n\}$.

- Das Zählen erfüllender Belegungen einer 2-KNF
wie auch das Zählen erfüllender Belegungen einer DNF,
- die Anzahl aller perfekten Matchings eines ungerichteten Graphen,
- die Anzahl aller k -Färbungen eines ungerichteten Graphen G .

Zusammenfassung

- In $DSPACE(o(\log_2 \log_2 n))$ berechenbare Sprachen stimmen mit den regulären Sprachen überein.
- Die erste nicht-triviale Klasse ist die Klasse DL aller auf logarithmischem Platz deterministisch berechenbaren Sprachen.
 - ▶ die Evaluierung von Polynomen, Dyck-Sprachen und **U-REACHABILITY** sind einige Sprachen in DL .
 - ▶ **D-REACHABILITY** ist NL -vollständig.
 - ★ Es gilt $DL \subseteq NL \subseteq P \subseteq NP$.
 - ★ Weitere NL -vollständige Probleme sind 2-SAT, das Leerheitsproblem für DFAs und die Simulation von NFAs.
- Der Satz von Savitch zeigt

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

sowie $NSPACE(s) \subseteq DSPACE(s^2)$ für platz-konstruierbare Funktionen s .

- Das Komplementverhalten ist „nicht typisch für Nichtdeterminismus“, denn

$$NSPACE(s) = coNSPACE(s)$$

gilt, falls s platz-konstruierbar ist.

- **PSPACE-Härte** bzw. **PSPACE-Vollständigkeit** ist eine Charakterisierung „richtig“ schwieriger Entscheidungsprobleme. Zu den Beispielen gehören:
 - ▶ Das Problem der Bestimmung einer Gewinnstrategie für nicht-triviale **Zwei-Personen Spiele** (wie QBF oder Geographie),
 - ▶ Entscheidungsprobleme für reguläre Ausdrücke oder NFA, wie **Universalität**, **das Äquivalenzproblem** oder **die Minimierung**.
- Später: PSPACE enthält alle Entscheidungsprobleme, die durch **randomisierte Algorithmen** oder **Quanten-Algorithmen** in polynomieller Zeit lösbar sind.
- Wir haben 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 $DSPACE(\log_2^2 n)$ lösbar.
 - ▶ Die Klasse der **regulären** Sprachen stimmt überein mit $DSPACE(s)$, falls $s = o(\log_2 \log_2 n)$.