

Theoretische Informatik II

Einheit 6.5

Hierarchie von Komplexitätsklassen



1. Pseudopolynomielle Algorithmen
2. Komplementäre Klassen
3. Polynomieller Platz
4. Hierarchiesätze

NICHT JEDES \mathcal{NP} -PROBLEM IST WIRKLICH SCHWER

Gibt es “leichte” \mathcal{NP} -vollständige Probleme?

NICHT JEDES \mathcal{NP} -PROBLEM IST WIRKLICH SCHWER

Gibt es “leichte” \mathcal{NP} -vollständige Probleme?

- **Was unterscheidet *CLIQUE* von *KP*?**
 - Beide Probleme sind \mathcal{NP} -vollständig, aber
 - $3SAT \leq_p CLIQUE$ codiert Formel durch gleich großen Graph
 - $3SAT \leq_p KP$ benutzt exponentiell große Zahlen als Codierung

NICHT JEDES \mathcal{NP} -PROBLEM IST WIRKLICH SCHWER

Gibt es “leichte” \mathcal{NP} -vollständige Probleme?

- **Was unterscheidet *CLIQUE* von *KP*?**
 - Beide Probleme sind \mathcal{NP} -vollständig, aber
 - $3SAT \leq_p CLIQUE$ codiert Formel durch gleich großen Graph
 - $3SAT \leq_p KP$ benutzt exponentiell große Zahlen als Codierung
 - Ist *KP* nur wegen der großen Zahlen \mathcal{NP} -vollständig?

NICHT JEDES \mathcal{NP} -PROBLEM IST WIRKLICH SCHWER

Gibt es “leichte” \mathcal{NP} -vollständige Probleme?

- **Was unterscheidet $CLIQUE$ von KP ?**

- Beide Probleme sind \mathcal{NP} -vollständig, aber
 - $3SAT \leq_p CLIQUE$ codiert Formel durch gleich großen Graph
 - $3SAT \leq_p KP$ benutzt exponentiell große Zahlen als Codierung
- Ist KP nur wegen der großen Zahlen \mathcal{NP} -vollständig?

- **Es gibt “bessere” Lösungen für KP**

$$KP = \{ (g_1..g_n, a_1..a_n, G, A) \mid \exists J \subseteq \{1..n\}. \sum_{i \in J} g_i \leq G \wedge \sum_{i \in J} a_i \geq A \}$$

- Man muß nicht alle Kombinationen von $\{1..n\}$ einzeln auswerten
- Man kann iterativ den optimalen Nutzen bestimmen,
indem man die Anzahl der Gegenstände und das Gewicht erhöht
- Sehr effizient, wenn das maximale Gewicht nicht zu groß wird

ITERATIVE LÖSUNG FÜR KP

$$KP = \{ (g_1..g_n, a_1..a_n, G, A) \mid \exists J \subseteq \{1..n\}. \sum_{i \in J} g_i \leq G \wedge \sum_{i \in J} a_i \geq A \}$$

ITERATIVE LÖSUNG FÜR KP

$$KP = \{ (g_1..g_n, a_1..a_n, G, A) \mid \exists J \subseteq \{1..n\}. \sum_{i \in J} g_i \leq G \wedge \sum_{i \in J} a_i \geq A \}$$

- **Betrachte Subprobleme $KP(k, g)$** (g und k fest)
 - Verwende Gegenstände $1, \dots, k$ und Maximalgewicht $g \leq G$
 - Definiere optimalen Nutzen $N(k, g)$
 - $N(k, 0) = 0$ für alle k
 - $N(0, g) = 0$ für alle g
 - $N(k, g) = \max\{N(k-1, g-g_k) + a_k, N(k-1, g)\}$

ITERATIVE LÖSUNG FÜR KP

$$KP = \{ (g_1..g_n, a_1..a_n, G, A) \mid \exists J \subseteq \{1..n\}. \sum_{i \in J} g_i \leq G \wedge \sum_{i \in J} a_i \geq A \}$$

- **Betrachte Subprobleme $KP(k, g)$** (g und k fest)
 - Verwende Gegenstände $1, \dots, k$ und Maximalgewicht $g \leq G$
 - Definiere optimalen Nutzen $N(k, g)$
 - $N(k, 0) = 0$ für alle k
 - $N(0, g) = 0$ für alle g
 - $N(k, g) = \max\{N(k-1, g-g_k) + a_k, N(k-1, g)\}$
- **Löse Rucksackproblem KP iterativ**
 - Es gilt $(g_1..g_n, a_1..a_n, G, A) \in KP \Leftrightarrow N(n, G) \geq A$
 - Gleichungen beschreiben rekursiven Algorithmus für $N(n, G)$
 - Tabellarischer Algorithmus bestimmt alle $N(k, g)$ mit $k \leq n, g \leq G$
 - Laufzeit ist $\mathcal{O}(n * G)$

ITERATIVE LÖSUNG FÜR KP

$$KP = \{ (g_1..g_n, a_1..a_n, G, A) \mid \exists J \subseteq \{1..n\}. \sum_{i \in J} g_i \leq G \wedge \sum_{i \in J} a_i \geq A \}$$

- **Betrachte Subprobleme $KP(k, g)$** (g und k fest)
 - Verwende Gegenstände $1, \dots, k$ und Maximalgewicht $g \leq G$
 - Definiere optimalen Nutzen $N(k, g)$
 - $N(k, 0) = 0$ für alle k
 - $N(0, g) = 0$ für alle g
 - $N(k, g) = \max\{N(k-1, g-g_k) + a_k, N(k-1, g)\}$
- **Löse Rucksackproblem KP iterativ**
 - Es gilt $(g_1..g_n, a_1..a_n, G, A) \in KP \Leftrightarrow N(n, G) \geq A$
 - Gleichungen beschreiben rekursiven Algorithmus für $N(n, G)$
 - Tabellarischer Algorithmus bestimmt alle $N(k, g)$ mit $k \leq n, g \leq G$
 - Laufzeit ist $\mathcal{O}(n * G)$

$(g_1..g_n, a_1..a_n, G, A) \in KP$ ist in $\mathcal{O}(n * G)$ Schritten lösbar

PSEUDOPOLYNOMIELLE ALGORITHMEN

Liegt das Rucksackproblem KP etwa in \mathcal{P} ?

PSEUDOPOLYNOMIELLE ALGORITHMEN

Liegt das Rucksackproblem KP etwa in \mathcal{P} ?

- **Lösung für KP ist nicht wirklich polynomiell**
 - $n * G$ kann exponentiell wachsen relativ zur Größe der Eingabe
 - Größe von $(g_1..g_n, a_1..a_n, G, A)$ ist $\mathcal{O}(n * (\log G + \log A))$

Liegt das Rucksackproblem KP etwa in \mathcal{P} ?

- **Lösung für KP ist nicht wirklich polynomiell**
 - $n * G$ kann exponentiell wachsen relativ zur Größe der Eingabe
 - Größe von $(g_1..g_n, a_1..a_n, G, A)$ ist $\mathcal{O}(n * (\log G + \log A))$
- **KP ist ein Zahlproblem**
 - $L \subseteq \Sigma^*$ ist **Zahlproblem**, wenn $MAX(w)$, die größte in einer Eingabe w codierte Zahl, nicht durch ein Polynom beschränkt werden kann

Liegt das Rucksackproblem KP etwa in \mathcal{P} ?

- **Lösung für KP ist nicht wirklich polynomiell**
 - $n * G$ kann exponentiell wachsen relativ zur Größe der Eingabe
 - Größe von $(g_1..g_n, a_1..a_n, G, A)$ ist $\mathcal{O}(n * (\log G + \log A))$
- **KP ist ein Zahlproblem**
 - $L \subseteq \Sigma^*$ ist **Zahlproblem**, wenn $MAX(w)$, die größte in einer Eingabe w codierte Zahl, nicht durch ein Polynom beschränkt werden kann
 - Weitere Zahlprobleme: $PARTITION, BPP, TSP, MSP, \dots$
 - Keine Zahlprobleme: $CLIQUE, VC, IS, SGI, LCS, DHC, HC, GC, \dots$

Liegt das Rucksackproblem KP etwa in \mathcal{P} ?

- **Lösung für KP ist nicht wirklich polynomiell**
 - $n * G$ kann exponentiell wachsen relativ zur Größe der Eingabe
 - Größe von $(g_1..g_n, a_1..a_n, G, A)$ ist $\mathcal{O}(n * (\log G + \log A))$
- **KP ist ein Zahlproblem**
 - $L \subseteq \Sigma^*$ ist **Zahlproblem**, wenn $MAX(w)$, die größte in einer Eingabe w codierte Zahl, nicht durch ein Polynom beschränkt werden kann
 - Weitere Zahlprobleme: *PARTITION, BPP, TSP, MSP, ...*
 - Keine Zahlprobleme: *CLIQUE, VC, IS, SGI, LCS, DHC, HC, GC, ...*
- **KP hat pseudopolynomielle Lösung**
 - Algorithmen für ein Zahlproblem $L \subseteq \Sigma^*$ sind **pseudopolynomiell**, wenn ihre Rechenzeit durch ein Polynom in $|w|$ und $MAX(w)$ beschränkt ist

STARKE \mathcal{NP} -VOLLSTÄNDIGKEIT

- **Pseudopolynomiell $\hat{=}$ effizient bei kleinen Zahlen**
 - Ist $L \subseteq \Sigma^*$ pseudopolynomiell lösbar, so ist für jedes Polynom p
 $L_p \equiv \{w \in L \mid \text{MAX}(w) \leq p(|w|)\} \in \mathcal{P}$
 - Die Restriktion von KP auf polynomiell große Gewichte liegt in \mathcal{P}

STARKE \mathcal{NP} -VOLLSTÄNDIGKEIT

- **Pseudopolynomiell $\hat{=}$ effizient bei kleinen Zahlen**

- Ist $L \subseteq \Sigma^*$ pseudopolynomiell lösbar, so ist für jedes Polynom p

$$L_p \equiv \{w \in L \mid \text{MAX}(w) \leq p(|w|)\} \in \mathcal{P}$$

- Die Restriktion von KP auf polynomiell große Gewichte liegt in \mathcal{P}

- Hat jedes Zahlproblem eine pseudopolynomielle Lösung?

STARKE \mathcal{NP} -VOLLSTÄNDIGKEIT

- **Pseudopolynomiell $\hat{=}$ effizient bei kleinen Zahlen**

- Ist $L \subseteq \Sigma^*$ pseudopolynomiell lösbar, so ist für jedes Polynom p

$$L_p \equiv \{w \in L \mid \text{MAX}(w) \leq p(|w|)\} \in \mathcal{P}$$

- Die Restriktion von KP auf polynomiell große Gewichte liegt in \mathcal{P}

- Hat jedes Zahlproblem eine pseudopolynomielle Lösung?

- **TSP ohne pseudopolynomielle Lösung**

(falls $\mathcal{P} \neq \mathcal{NP}$)

- Der Reduktionsbeweis $HC \leq_p TSP$ zeigt $HC \leq_p TSP_n$

- Eine Restriktion von TSP auf kleine Zahlen bleibt \mathcal{NP} -vollständig

STARKE \mathcal{NP} -VOLLSTÄNDIGKEIT

- **Pseudopolynomiell $\hat{=}$ effizient bei kleinen Zahlen**

- Ist $L \subseteq \Sigma^*$ pseudopolynomiell lösbar, so ist für jedes Polynom p

$$L_p \equiv \{w \in L \mid \text{MAX}(w) \leq p(|w|)\} \in \mathcal{P}$$

- Die Restriktion von KP auf polynomiell große Gewichte liegt in \mathcal{P}

- Hat jedes Zahlproblem eine pseudopolynomielle Lösung?

- **TSP ohne pseudopolynomielle Lösung**

(falls $\mathcal{P} \neq \mathcal{NP}$)

- Der Reduktionsbeweis $HC \leq_p TSP$ zeigt $HC \leq_p TSP_n$

- Eine Restriktion von TSP auf kleine Zahlen bleibt \mathcal{NP} -vollständig

- **TSP ist stark \mathcal{NP} -vollständig**

- $L \subseteq \Sigma^*$ **stark \mathcal{NP} -vollständig** $\equiv L_p$ \mathcal{NP} -vollständig für ein Polynom p

- L stark \mathcal{NP} -vollständig $\Rightarrow L$ hat keine pseudopolynomielle Lösung

ES GIBT WEITERE WICHTIGE KOMPLEXITÄTSKLASSEN

- *co-NP*
 - Probleme mit Komplement in \mathcal{NP}
 - Problem muß nicht notwendigerweise selbst in \mathcal{NP} liegen

ES GIBT WEITERE WICHTIGE KOMPLEXITÄTSKLASSEN

- *co-NP*
 - Probleme mit Komplement in \mathcal{NP}
 - Problem muß nicht notwendigerweise selbst in \mathcal{NP} liegen
- *PSPACE*
 - Platzverbrauch polynomiell relativ zur Größe der Eingabe

ES GIBT WEITERE WICHTIGE KOMPLEXITÄTSKLASSEN

- ***co-NP***
 - Probleme mit Komplement in \mathcal{NP}
 - Problem muß nicht notwendigerweise selbst in \mathcal{NP} liegen
- ***PSPACE***
 - Platzverbrauch polynomiell relativ zur Größe der Eingabe
- **Klassen mit großer theoretischer Bedeutung**
 - Σ_2^P : Sprachen von OTMs, deren Orakel ein \mathcal{NP} -Problem entscheidet
 - Π_2^P : Sprachen von OTMs, deren Orakel ein $co\text{-}\mathcal{NP}$ -Problem entscheidet
 - Σ_i^P / Π_i^P : Sprachen von OTMs mit Orakel in $\Pi_{i-1}^P / \Sigma_{i-1}^P$
 - ***LOGSPACE***: logarithmischer Platzverbrauch der Berechnung (!)
 - ***EXPTIME / EXPSPACE***: exponentieller Zeit-/Platzbedarf
Rechenzeit und Platzverbrauch sind nicht mehr handhabbar

co-NP: PROBLEME MIT KOMPLEMENT IN \mathcal{NP}

$$\textit{co-C} := \{ L \mid \bar{L} \in \mathcal{C} \}$$

$co-\mathcal{NP}$: PROBLEME MIT KOMPLEMENT IN \mathcal{NP}

$$co-\mathcal{C} := \{ L \mid \bar{L} \in \mathcal{C} \}$$

- **Interessant für nichtdeterministische Klassen**
 - Für deterministische Komplexitätsklassen gilt $\mathcal{C} = co-\mathcal{C}$
Akzeptieren/Verwerfen einer DTM ist vertauschbar
 - Nichtdeterministisches Akzeptieren ist komplizierter:
OTM akzeptiert, wenn Orakel **einen** akzeptablen Lösungsvorschlag machen kann

$co-\mathcal{NP}$: PROBLEME MIT KOMPLEMENT IN \mathcal{NP}

$$co-\mathcal{C} := \{ L \mid \bar{L} \in \mathcal{C} \}$$

- **Interessant für nichtdeterministische Klassen**
 - Für deterministische Komplexitätsklassen gilt $\mathcal{C} = co-\mathcal{C}$
Akzeptieren/Verwerfen einer DTM ist vertauschbar
 - Nichtdeterministisches Akzeptieren ist komplizierter:
OTM akzeptiert, wenn Orakel **einen** akzeptablen Lösungsvorschlag machen kann
- **Beispiele für Probleme in $co-\mathcal{NP}$:**
 - Menge der allgemeingültigen Formeln (Komplement von SAT)
 - Das Primzahlproblem (Komplement von Zusammengesetztheit)

$co-\mathcal{NP}$: PROBLEME MIT KOMPLEMENT IN \mathcal{NP}

$$co-\mathcal{C} := \{ L \mid \bar{L} \in \mathcal{C} \}$$

- **Interessant für nichtdeterministische Klassen**
 - Für deterministische Komplexitätsklassen gilt $\mathcal{C} = co-\mathcal{C}$
Akzeptieren/Verwerfen einer DTM ist vertauschbar
 - Nichtdeterministisches Akzeptieren ist komplizierter:
OTM akzeptiert, wenn Orakel **einen** akzeptablen Lösungsvorschlag machen kann
- **Beispiele für Probleme in $co-\mathcal{NP}$:**
 - Menge der allgemeingültigen Formeln (Komplement von SAT)
 - Das Primzahlproblem (Komplement von Zusammengesetztheit)
- **Bisherige Erkenntnisse deuten auf $co-\mathcal{NP} \neq \mathcal{NP}$**
 - Wenn $\mathcal{P} = \mathcal{NP}$, dann $co-\mathcal{NP} = co-\mathcal{P} = \mathcal{P} = \mathcal{NP}$

$co-\mathcal{NP}$: PROBLEME MIT KOMPLEMENT IN \mathcal{NP}

$$co-\mathcal{C} := \{ L \mid \bar{L} \in \mathcal{C} \}$$

- **Interessant für nichtdeterministische Klassen**

- Für deterministische Komplexitätsklassen gilt $\mathcal{C} = co-\mathcal{C}$
Akzeptieren/Verwerfen einer DTM ist vertauschbar

- Nichtdeterministisches Akzeptieren ist komplizierter:
OTM akzeptiert, wenn Orakel **einen** akzeptablen Lösungsvorschlag machen kann

- **Beispiele für Probleme in $co-\mathcal{NP}$:**

- Menge der allgemeingültigen Formeln (Komplement von SAT)
- Das Primzahlproblem (Komplement von Zusammengesetztheit)

- **Bisherige Erkenntnisse deuten auf $co-\mathcal{NP} \neq \mathcal{NP}$**

- Wenn $\mathcal{P} = \mathcal{NP}$, dann $co-\mathcal{NP} = co-\mathcal{P} = \mathcal{P} = \mathcal{NP}$

- $\mathcal{NP} = co-\mathcal{NP} \Leftrightarrow \mathcal{NPC} \cap co-\mathcal{NP} \neq \emptyset$

- \Rightarrow : offensichtlich, da $\mathcal{NPC} \neq \emptyset$

- \Leftarrow : Ist $L \in \mathcal{NPC} \cap co-\mathcal{NP}$ so gilt $\bar{L}' \leq_p L$ für jedes $L' \in co-\mathcal{NP}$ also $L' \leq_p \bar{L} \in \mathcal{NP}$

PSPACE: POLYNOMIELLER PLATZVERBRAUCH

- $\mathcal{NP} \subseteq PSPACE$

- Eine Maschine kann in polynomieller Zeit nur polynomiell viele Speicherzellen verwenden

PSPACE: POLYNOMIELLER PLATZVERBRAUCH

- $\mathcal{NP} \subseteq PSPACE$

- Eine Maschine kann in polynomieller Zeit nur polynomiell viele Speicherzellen verwenden

- $NSPACE(f(n)) \subseteq SPACE(f(n)^2)$ (Satz von Savitch)

- Simulation mit Speicherung der Alternativen (§4.1) zu platzaufwendig

PSPACE: POLYNOMIELLER PLATZVERBRAUCH

- $\mathcal{NP} \subseteq PSPACE$

- Eine Maschine kann in polynomieller Zeit nur polynomiell viele Speicherzellen verwenden

- $NSPACE(f(n)) \subseteq SPACE(f(n)^2)$ (Satz von Savitch)

- Simulation mit Speicherung der Alternativen (§4.1) zu platzaufwendig
- Teste $\kappa_\alpha \vdash^t \kappa_\omega$ (für maximales $t=2^{c \cdot f(n)}$) durch “binäre Tiefensuche”
- Platzverbrauch des Rekursionsstacks ist $\mathcal{O}(f(n)^2)$ (falls $f(n) \geq \log n$)
- Zeitaufwand der Simulation ist exponentiell höher als bei der NTM

PSPACE: POLYNOMIELLER PLATZVERBRAUCH

- $\mathcal{NP} \subseteq PSPACE$

- Eine Maschine kann in polynomieller Zeit nur polynomiell viele Speicherzellen verwenden

- $NSPACE(f(n)) \subseteq SPACE(f(n)^2)$ (Satz von Savitch)

- Simulation mit Speicherung der Alternativen (§4.1) zu platzaufwendig
- Teste $\kappa_\alpha \vdash^t \kappa_\omega$ (für maximales $t=2^{c \cdot f(n)}$) durch “binäre Tiefensuche”
- Platzverbrauch des Rekursionsstacks ist $\mathcal{O}(f(n)^2)$ (falls $f(n) \geq \log n$)
- Zeitaufwand der Simulation ist exponentiell höher als bei der NTM

- $PSPACE = NPSPACE$

HMU §11.2.3

- Folgt direkt aus dem Satz von Savitch

PSPACE: POLYNOMIELLER PLATZVERBRAUCH

- *NP* \subseteq *PSPACE*

- Eine Maschine kann in polynomieller Zeit nur polynomiell viele Speicherzellen verwenden

- *NSPACE*($f(n)$) \subseteq *SPACE*($f(n)^2$) (Satz von Savitch)

- Simulation mit Speicherung der Alternativen (§4.1) zu platzaufwendig
- Teste $\kappa_\alpha \vdash^t \kappa_\omega$ (für maximales $t=2^{c \cdot f(n)}$) durch “binäre Tiefensuche”
- Platzverbrauch des Rekursionsstacks ist $\mathcal{O}(f(n)^2)$ (falls $f(n) \geq \log n$)
- Zeitaufwand der Simulation ist exponentiell höher als bei der NTM

- *PSPACE* = *NPSPACE*

HMU §11.2.3

- Folgt direkt aus dem Satz von Savitch

- *PSPACE* \subseteq *EXPTIME*

- Eine terminierende Maschine, die maximal $f(n)$ Bandzellen aufsucht, terminiert nach maximal $|\Gamma|^{f(n)} * |Q| * f(n)$ Schritten
- Zahl der Konfigurationen = Bandinhalte * Zustände * Kopfpositionen

NPSPACE \subseteq *NEXPTIME* gilt aus dem gleichen Grund

PSPACE-VOLLSTÄNDIGKEIT

- **\mathcal{C} -Vollständigkeit allgemein**
 - L ist \mathcal{C} -vollständig, falls $L \in \mathcal{C}$ und $L' \leq_p L$ für alle $L' \in \mathcal{C}$

PSPACE-VOLLSTÄNDIGKEIT

- ***C*-Vollständigkeit allgemein**
 - *L* ist *C*-vollständig, falls $L \in \mathcal{C}$ und $L' \leq_p L$ für alle $L' \in \mathcal{C}$
- **Wie zeigt man *PSPACE*-Vollständigkeit?**
 - Codiere Berechnungen von DTMs, die polynomiellen Platz brauchen
 - Sprache: komplexer als *SAT*, aber mit deterministischer Natur
 - Kandidat: Wahrheit geschlossener quantifizierter boolescher Formeln

PSPACE-VOLLSTÄNDIGKEIT

- **\mathcal{C} -Vollständigkeit allgemein**

- L ist \mathcal{C} -vollständig, falls $L \in \mathcal{C}$ und $L' \leq_p L$ für alle $L' \in \mathcal{C}$

- **Wie zeigt man PSPACE-Vollständigkeit?**

- Codiere Berechnungen von DTMs, die polynomiellen Platz brauchen
- Sprache: komplexer als SAT , aber mit deterministischer Natur
- Kandidat: Wahrheit geschlossener quantifizierter boolescher Formeln

- **Erweitere Aussagenlogik um boolesche Quantoren**

- Aussagenlogische Variablen P, Q, R, \dots , Konstante t und f
- Formeln $\neg F, E \wedge F, E \vee F, E \Rightarrow F, (\forall P)F, (\exists P)F$
- Wert von $(\forall P)F$ entspricht dem von $F[t/P] \wedge F[f/P]$
- Wert von $(\exists P)F$ entspricht dem von $F[t/P] \vee F[f/P]$
- Wert anderer Formeln wie in gewöhnlicher Aussagenlogik

PSPACE-VOLLSTÄNDIGKEIT

- **C-Vollständigkeit allgemein**

- L ist **C-vollständig**, falls $L \in \mathcal{C}$ und $L' \leq_p L$ für alle $L' \in \mathcal{C}$

- **Wie zeigt man PSPACE-Vollständigkeit?**

- Codiere Berechnungen von DTMs, die polynomiellen Platz brauchen
- Sprache: komplexer als SAT , aber mit deterministischer Natur
- Kandidat: **Wahrheit geschlossener quantifizierter boolescher Formeln**

- **Erweitere Aussagenlogik um boolesche Quantoren**

- Aussagenlogische Variablen P, Q, R, \dots , Konstante t und f
- Formeln $\neg F, E \wedge F, E \vee F, E \Rightarrow F, (\forall P)F, (\exists P)F$
- Wert von $(\forall P)F$ entspricht dem von $F[t/P] \wedge F[f/P]$
- Wert von $(\exists P)F$ entspricht dem von $F[t/P] \vee F[f/P]$
- Wert anderer Formeln wie in gewöhnlicher Aussagenlogik
- $(\forall P)(\exists Q) [(P \vee Q) \wedge (\neg P \vee \neg Q)]$ ist wahr (Wert 1)
- $(\exists Q)(\forall P) [(P \vee Q) \wedge (\neg P \vee \neg Q)]$ ist falsch (Wert 0)

PSPACE-VOLLSTÄNDIGKEIT

- **C-Vollständigkeit allgemein**

- L ist **C-vollständig**, falls $L \in \mathcal{C}$ und $L' \leq_p L$ für alle $L' \in \mathcal{C}$

- **Wie zeigt man PSPACE-Vollständigkeit?**

- Codiere Berechnungen von DTMs, die polynomiellen Platz brauchen
- Sprache: komplexer als SAT , aber mit deterministischer Natur
- Kandidat: **Wahrheit geschlossener quantifizierter boolescher Formeln**

- **Erweitere Aussagenlogik um boolesche Quantoren**

- Aussagenlogische Variablen P, Q, R, \dots , Konstante t und f
- Formeln $\neg F, E \wedge F, E \vee F, E \Rightarrow F, (\forall P)F, (\exists P)F$
- Wert von $(\forall P)F$ entspricht dem von $F[t/P] \wedge F[f/P]$
- Wert von $(\exists P)F$ entspricht dem von $F[t/P] \vee F[f/P]$
- Wert anderer Formeln wie in gewöhnlicher Aussagenlogik
- $(\forall P)(\exists Q) [(P \vee Q) \wedge (\neg P \vee \neg Q)]$ ist wahr (Wert 1)
- $(\exists Q)(\forall P) [(P \vee Q) \wedge (\neg P \vee \neg Q)]$ ist falsch (Wert 0)

QBF ist die Menge der geschlossenen QB-Formeln mit Wert 1

QBF IST *PSPACE*-VOLLSTÄNDIG

- *QBF* ∈ *PSPACE*

- Auswerten aussagenlogischer Formeln braucht linearen Platz

- $(\forall P)F = F[t/P] \wedge F[f/P]$ und $(\exists P)F = F[t/P] \vee F[f/P]$
werden kaskadisch ausgewertet

- Gesamtbedarf, einschließlich Zwischenspeicherung, ist quadratisch

QBF IST PSPACE-VOLLSTÄNDIG

- **QBF \in PSPACE**

- Auswerten aussagenlogischer Formeln braucht linearen Platz
- $(\forall P)F = F[t/P] \wedge F[f/P]$ und $(\exists P)F = F[t/P] \vee F[f/P]$ werden kaskadisch ausgewertet
- Gesamtbedarf, einschließlich Zwischenspeicherung, ist quadratisch

- **Für alle $L \in PSPACE$ gilt $L \leq_p QBF$** HMU §11.3.4

- Codiere Bandzellen und Konfigurationen wie im Satz von Cook
- Beschreibe Formeln $F_{\kappa_1, \kappa_2, t}$ für die Aussage $\kappa_1 \vdash^t \kappa_2$
- Zielformel ist $F_{\kappa_\alpha, \kappa_\omega, 2^{c \cdot p(n)}}$, wobei κ_α Anfangskonfiguration, κ_ω Endkonfiguration, $p(n)$ Platzverbrauch, c Alternativen pro Zelle

QBF IST PSPACE-VOLLSTÄNDIG

• $QBF \in PSPACE$

- Auswerten aussagenlogischer Formeln braucht linearen Platz
- $(\forall P)F = F[t/P] \wedge F[f/P]$ und $(\exists P)F = F[t/P] \vee F[f/P]$ werden kaskadisch ausgewertet
- Gesamtbedarf, einschließlich Zwischenspeicherung, ist quadratisch

• Für alle $L \in PSPACE$ gilt $L \leq_p QBF$ HMU §11.3.4

- Codiere Bandzellen und Konfigurationen wie im Satz von Cook
 - Beschreibe Formeln $F_{\kappa_1, \kappa_2, t}$ für die Aussage $\kappa_1 \vdash^t \kappa_2$
 - Zielformel ist $F_{\kappa_\alpha, \kappa_\omega, 2^{c \cdot p(n)}}$, wobei κ_α Anfangskonfiguration, κ_ω Endkonfiguration, $p(n)$ Platzverbrauch, c Alternativen pro Zelle
 - Setze $F_{\kappa_1, \kappa_1, 0}$ und beschreibe $F_{\kappa_1, \kappa_2, 1}$ passend zur Tabelle von δ
 - Beschreibe $F_{\kappa_1, \kappa_2, t}$ durch eine Darstellung für $(\exists \kappa) F_{\kappa_1, \kappa, t \div 2} \wedge F_{\kappa, \kappa_2, t \div 2}$, die das Entstehen exponentiell großer Formeln vermeidet
- $$(\exists \kappa)(\forall \kappa_3)(\forall \kappa_4)[(\kappa_3 \Leftrightarrow \kappa_1 \wedge \kappa_4 \Leftrightarrow \kappa) \vee (\kappa_3 \Leftrightarrow \kappa \wedge \kappa_4 \Leftrightarrow \kappa_2)] \Rightarrow F_{\kappa_3, \kappa_4, t \div 2}$$

- **Strategische 2-Personen Spiele**

- Viele konkrete Beispiele in [Garey/Johnson Seite 254ff](#)
- Spielentscheidungen entsprechen alternierenden QB Formeln
Spieler gewinnt, wenn für jeden Zug des Gegners, ein Zug existiert, so daß für jeden Folgezug des Gegners, ... das Resultat einen Sieg darstellt
- QBF kann als strategisches Spiel beschrieben werden (und umgekehrt)

- **Strategische 2-Personen Spiele**

- Viele konkrete Beispiele in [Garey/Johnson Seite 254ff](#)
- Spielentscheidungen entsprechen alternierenden QB Formeln
Spieler gewinnt, wenn für jeden Zug des Gegners, ein Zug existiert, so daß für jeden Folgezug des Gegners, ... das Resultat einen Sieg darstellt
- QBF kann als strategisches Spiel beschrieben werden (und umgekehrt)

- **Sprache regulärer Ausdrücke**

- Ist $L(E) = \Sigma^*$ für einen beliebigen regulären Ausdruck über Σ ?

WEITERE *PSPACE*-VOLLSTÄNDIGE PROBLEME

- **Strategische 2-Personen Spiele**

- Viele konkrete Beispiele in [Garey/Johnson Seite 254ff](#)
- Spielentscheidungen entsprechen alternierenden QB Formeln
Spieler gewinnt, wenn für jeden Zug des Gegners, ein Zug existiert, so daß für jeden Folgezug des Gegners, ... das Resultat einen Sieg darstellt
- QBF kann als strategisches Spiel beschrieben werden (und umgekehrt)

- **Sprache regulärer Ausdrücke**

- Ist $L(E) = \Sigma^*$ für einen beliebigen regulären Ausdruck über Σ ?

- **In-Place Acceptance**

[Asteroth/Baier §4.5](#)

- Kann eine gegebene DTM jedes Wort w ihrer Sprache mit Platzbedarf $|w|$ akzeptieren?

WICHTIGE VERTRETER ANDERER KOMPLEXITÄTSKLASSEN

- **Isomorphie ungerichteter Graphen**

\mathcal{NP} , nicht vollständig

WICHTIGE VERTRETER ANDERER KOMPLEXITÄTSKLASSEN

- **Isomorphie ungerichteter Graphen** \mathcal{NP} , nicht vollständig
- **Zuverlässigkeit von Netzwerken** \mathcal{NP} -hart, vermutlich nicht in \mathcal{NP}
 - Wahrscheinlichkeit für fehlerfreie Verbindung zwischen zwei Knoten

WICHTIGE VERTRETER ANDERER KOMPLEXITÄTSKLASSEN

- **Isomorphie ungerichteter Graphen** \mathcal{NP} , nicht vollständig
- **Zuverlässigkeit von Netzwerken** \mathcal{NP} -hart, vermutlich nicht in \mathcal{NP}
 - Wahrscheinlichkeit für fehlerfreie Verbindung zwischen zwei Knoten
- **Minimale äquivalente Schaltkreise** Σ_2^P , also \mathcal{NP} -hart, nicht in \mathcal{NP}
 - Bestimme optimale Größe einer Schaltung

WICHTIGE VERTRETER ANDERER KOMPLEXITÄTSKLASSEN

- **Isomorphie ungerichteter Graphen** \mathcal{NP} , nicht vollständig
- **Zuverlässigkeit von Netzwerken** \mathcal{NP} -hart, vermutlich nicht in \mathcal{NP}
 - Wahrscheinlichkeit für fehlerfreie Verbindung zwischen zwei Knoten
- **Minimale äquivalente Schaltkreise** Σ_2^P , also \mathcal{NP} -hart, nicht in \mathcal{NP}
 - Bestimme optimale Größe einer Schaltung
- **$n \times n$ -Schach, Dame, Go** $EXPTIME$ (-vollständig)
 - Exponentiell viele Züge bis Spielende möglich

WICHTIGE VERTRETER ANDERER KOMPLEXITÄTSKLASSEN

- **Isomorphie ungerichteter Graphen** \mathcal{NP} , nicht vollständig
- **Zuverlässigkeit von Netzwerken** \mathcal{NP} -hart, vermutlich nicht in \mathcal{NP}
 - Wahrscheinlichkeit für fehlerfreie Verbindung zwischen zwei Knoten
- **Minimale äquivalente Schaltkreise** Σ_2^P , also \mathcal{NP} -hart, nicht in \mathcal{NP}
 - Bestimme optimale Größe einer Schaltung
- **$n \times n$ -Schach, Dame, Go** $EXPTIME$ (-vollständig)
 - Exponentiell viele Züge bis Spielende möglich
- **TSP***: Bestimmung **aller** Rundreisen mit gegebenen Kosten $EXSPACE$
 - Unrealistische Problemstellung: zu viele Lösungen

WICHTIGE VERTRETER ANDERER KOMPLEXITÄTSKLASSEN

- **Isomorphie ungerichteter Graphen** \mathcal{NP} , nicht vollständig
- **Zuverlässigkeit von Netzwerken** \mathcal{NP} -hart, vermutlich nicht in \mathcal{NP}
 - Wahrscheinlichkeit für fehlerfreie Verbindung zwischen zwei Knoten
- **Minimale äquivalente Schaltkreise** Σ_2^P , also \mathcal{NP} -hart, nicht in \mathcal{NP}
 - Bestimme optimale Größe einer Schaltung
- **$n \times n$ -Schach, Dame, Go** $EXPTIME$ (-vollständig)
 - Exponentiell viele Züge bis Spielende möglich
- **TSP***: Bestimmung **aller** Rundreisen mit gegebenen Kosten $EXPSPACE$
 - Unrealistische Problemstellung: zu viele Lösungen
- **Äquivalenz regulärer Ausdrücke mit Iteration** $EXPSPACE$ -vollständig
 - Einfache Problemstellung mit sehr schwieriger Lösung
 - Ausdrücke dürfen $E^k = \underbrace{E \circ E \dots \circ E}_{k\text{-mal}}$ enthalten

FÜHRT MEHR ZEIT/PLATZ ZU MEHR LÖSBAREN PROBLEMEN?

- **Welche der folgenden Inklusionen sind echt?**

$LOGSPACE \subseteq NLOGSPACE$

$\subseteq \mathcal{P} \subseteq \mathcal{NP} \subseteq PSPACE = NPSPACE$

$\subseteq EXPTIME \subseteq NEXPTIME \subseteq EXPSPACE \subseteq \dots$

FÜHRT MEHR ZEIT/PLATZ ZU MEHR LÖSBAREN PROBLEMEN?

- **Welche der folgenden Inklusionen sind echt?**

$LOGSPACE \subseteq NLOGSPACE$

$\subseteq \mathcal{P} \subseteq \mathcal{NP} \subseteq PSPACE = NPSPACE$

$\subseteq EXPTIME \subseteq NEXPTIME \subseteq EXPSPACE \subseteq \dots$

- **Wie beweist man Unlösbarkeit in Platz/Zeit f**

– Diagonalisierung über alle Probleme, die in Komplexität f lösbar sind

FÜHRT MEHR ZEIT/PLATZ ZU MEHR LÖSBAREN PROBLEMEN?

- **Welche der folgenden Inklusionen sind echt?**

$LOGSPACE \subseteq NLOGSPACE$
 $\subseteq \mathcal{P} \subseteq \mathcal{NP} \subseteq PSPACE = NPSPACE$
 $\subseteq EXPTIME \subseteq NEXPTIME \subseteq EXPSPACE \subseteq \dots$

- **Wie beweist man Unlösbarkeit in Platz/Zeit f**

– Diagonalisierung über alle Probleme, die in Komplexität f lösbar sind

- **Hilfsmittel: Konstruierbare Funktionen**

– Funktionen, deren Komplexität durch ihren Wert beschränkt ist

FÜHRT MEHR ZEIT/PLATZ ZU MEHR LÖSBAREN PROBLEMEN?

- **Welche der folgenden Inklusionen sind echt?**

$LOGSPACE \subseteq NLOGSPACE$
 $\subseteq \mathcal{P} \subseteq \mathcal{NP} \subseteq PSPACE = NPSPACE$
 $\subseteq EXPTIME \subseteq NEXPTIME \subseteq EXPSPACE \subseteq \dots$

- **Wie beweist man Unlösbarkeit in Platz/Zeit f**

– Diagonalisierung über alle Probleme, die in Komplexität f lösbar sind

- **Hilfsmittel: Konstruierbare Funktionen**

– Funktionen, deren Komplexität durch ihren Wert beschränkt ist

– $f : \mathbb{N} \rightarrow \mathbb{N}$ ist **platzkonstruierbar**, wenn \hat{f} in Platz $\mathcal{O}(f)$ berechenbar

– $f : \mathbb{N} \rightarrow \mathbb{N}$ ist **zeitkonstruierbar**, wenn \hat{f} in Zeit $\mathcal{O}(f)$ berechenbar

$\hat{f} : \{1\}^* \rightarrow \{0, 1, \}^*$ berechnet bei Eingabe 1^n die Binärdarstellung $r_b(f(n))$ von $f(n)$

Rahmenbedingungen: $f(n) \geq \log n$ (Platz) bzw. $f(n) \geq n \log n$ (Zeit) für alle n

– $\log n, n^k, 2^n$ etc sind zeit- und platzkonstruierbar

FÜHRT MEHR ZEIT/PLATZ ZU MEHR LÖSBAREN PROBLEMEN?

- **Welche der folgenden Inklusionen sind echt?**

$LOGSPACE \subseteq NLOGSPACE$
 $\subseteq \mathcal{P} \subseteq \mathcal{NP} \subseteq PSPACE = NPSPACE$
 $\subseteq EXPTIME \subseteq NEXPTIME \subseteq EXPSPACE \subseteq \dots$

- **Wie beweist man Unlösbarkeit in Platz/Zeit f**

– Diagonalisierung über alle Probleme, die in Komplexität f lösbar sind

- **Hilfsmittel: Konstruierbare Funktionen**

– Funktionen, deren Komplexität durch ihren Wert beschränkt ist

– $f : \mathbb{N} \rightarrow \mathbb{N}$ ist **platzkonstruierbar**, wenn \hat{f} in Platz $\mathcal{O}(f)$ berechenbar

– $f : \mathbb{N} \rightarrow \mathbb{N}$ ist **zeitkonstruierbar**, wenn \hat{f} in Zeit $\mathcal{O}(f)$ berechenbar

$\hat{f} : \{1\}^* \rightarrow \{0, 1, \}^*$ berechnet bei Eingabe 1^n die Binärdarstellung $r_b(f(n))$ von $f(n)$

Rahmenbedingungen: $f(n) \geq \log n$ (Platz) bzw. $f(n) \geq n \log n$ (Zeit) für alle n

– $\log n, n^k, 2^n$ etc sind zeit- und platzkonstruierbar

- **Wichtiger formaler Begriff: Ordnung $o(f)$**

– f als echte obere Schranke: $o(f) = \{g : \mathbb{N} \rightarrow \mathbb{R}^+ \mid \forall c > 0. g <_a c * f\}$

DAS PLATZHIERARCHIE-THEOREM

Für jede platzkonstruierbare Funktion f gibt es eine Sprache $L \in SPACE(f)$, deren Platzkomplexität nicht in $o(f)$ liegt

DAS PLATZHIERARCHIE-THEOREM

Für jede platzkonstruierbare Funktion f gibt es eine Sprache $L \in SPACE(f)$, deren Platzkomplexität nicht in $o(f)$ liegt

- **Konstruiere L durch Diagonalisierung**
 - Definiere L durch Konstruktion ihrer charakteristischen Funktion

DAS PLATZHIERARCHIE-THEOREM

Für jede platzkonstruierbare Funktion f gibt es eine Sprache $L \in SPACE(f)$, deren Platzkomplexität nicht in $o(f)$ liegt

- **Konstruiere L durch Diagonalisierung**

- Definiere L durch Konstruktion ihrer charakteristischen Funktion

- Sei $h(n) = \begin{cases} 1 & \Phi_i(n) \leq 2^{f(n)} \wedge s_{M_i}(n) \leq^{**} f(n) \wedge \varphi_i(n) = 0 \quad \text{mit } i = \pi_1(n) \\ 0 & \text{sonst} \end{cases}$

$s_{M_i}(n) \leq^{**} f(n) \equiv h$ benutzt zur Simulation von $\varphi_i(n)$ maximal Platz $f(n)$.

Benutzt die Simulation von $\varphi_i(n)$ Platz $d \cdot s_{M_i}(n)$, so muß $s_{M_i}(n) \leq f(n)/d$ gelten

- h ist eine Entscheidungsfunktion, die in Platz $\mathcal{O}(f)$ berechenbar ist

- Definiere $L := h^{-1}(\{1\})$ (also $\chi_L = h$), also $L \in SPACE(f)$

DAS PLATZHIERARCHIE-THEOREM

Für jede platzkonstruierbare Funktion f gibt es eine Sprache $L \in SPACE(f)$, deren Platzkomplexität nicht in $o(f)$ liegt

- **Konstruiere L durch Diagonalisierung**

- Definiere L durch Konstruktion ihrer charakteristischen Funktion

- Sei $h(n) = \begin{cases} 1 & \Phi_i(n) \leq 2^{f(n)} \wedge s_{M_i}(n) \leq^{**} f(n) \wedge \varphi_i(n) = 0 \quad \text{mit } i = \pi_1(n) \\ 0 & \text{sonst} \end{cases}$

$s_{M_i}(n) \leq^{**} f(n) \equiv h$ benutzt zur Simulation von $\varphi_i(n)$ maximal Platz $f(n)$.

Benutzt die Simulation von $\varphi_i(n)$ Platz $d * s_{M_i}(n)$, so muß $s_{M_i}(n) \leq f(n)/d$ gelten

- h ist eine Entscheidungsfunktion, die in Platz $\mathcal{O}(f)$ berechenbar ist

- Definiere $L := h^{-1}(\{1\})$ (also $\chi_L = h$), also $L \in SPACE(f)$

- **Die Platzkomplexität von L ist nicht in $o(f)$**

- Falls L durch ein Programm mit Komplexität $o(f)$ entschieden wird, so gilt $\chi_L = \varphi_k$ für ein k mit $s_{M_k}(n) < c * f(n)$ für alle $c > 0$, $n \geq n_0$

- Wähle $n := \langle k, n_0 \rangle$ (also $k = \pi_1(n)$) für das zu $(1/d)$ passende n_0

Dann gilt $n \geq n_0$, also $s_{M_k}(n) < (1/d) * f(n)$ bzw $s_{M_i}(n) \leq^{**} f(n)$

- Es folgt $n \in L \Leftrightarrow h(n) = 1 \Leftrightarrow \varphi_k(n) = 0 \wedge s_{M_i}(n) \leq^{**} f(n) \Leftrightarrow n \notin L$

KONSEQUENZEN DES HIERARCHIETHEOREMS

- **Platzkomplexität bildet eine echte Hierarchie**
 - $SPACE(f) \subset SPACE(g)$ falls g platzkonstruierbar und $f \in o(g)$
 - $SPACE(n^\epsilon) \subset SPACE(n^{\epsilon'})$ für alle $0 \leq \epsilon < \epsilon'$
 - $NLOGSPACE \subseteq SPACE(\log^2 n) \subset SPACE(n) \subset PSPACE$
 - $NPSPACE \subset EXPSPACE$

KONSEQUENZEN DES HIERARCHIETHEOREMS

- **Platzkomplexität bildet eine echte Hierarchie**

- $SPACE(f) \subset SPACE(g)$ falls g platzkonstruierbar und $f \in o(g)$
- $SPACE(n^\epsilon) \subset SPACE(n^{\epsilon'})$ für alle $0 \leq \epsilon < \epsilon'$
- $NLOGSPACE \subseteq SPACE(\log^2 n) \subset SPACE(n) \subset PSPACE$
- $NPSPACE \subset EXPSPACE$

- **Zeitkomplexität bildet eine echte Hierarchie**

- **Für jede zeitkonstruierbare Funktion f gibt es eine mSprache $L \in TIME(f)$ deren Zeitkomplexität nicht in $o(f / \log f)$ liegt**
 - Beweis analog zu Platzhierarchietheorem

KONSEQUENZEN DES HIERARCHIETHEOREMS

- **Platzkomplexität bildet eine echte Hierarchie**

- $SPACE(f) \subset SPACE(g)$ falls g platzkonstruierbar und $f \in o(g)$
- $SPACE(n^\epsilon) \subset SPACE(n^{\epsilon'})$ für alle $0 \leq \epsilon < \epsilon'$
- $NLOGSPACE \subseteq SPACE(\log^2 n) \subset SPACE(n) \subset PSPACE$
- $NPSPACE \subset EXPSPACE$

- **Zeitkomplexität bildet eine echte Hierarchie**

- **Für jede zeitkonstruierbare Funktion f gibt es eine mSprache $L \in TIME(f)$ deren Zeitkomplexität nicht in $o(f / \log f)$ liegt**
 - Beweis analog zu Platzhierarchietheorem
- $TIME(f) \subset TIME(g)$ falls g platzkonstruierbar und $f \in o(g / \log g)$
- $TIME(n^\epsilon) \subset TIME(n^{\epsilon'})$ für alle $0 \leq \epsilon < \epsilon'$
- $\mathcal{P} \subset EXPTIME$

KOMPLEXITÄTSKLASSENHIERARCHIE

