

# Theoretische Informatik und Logik

## 12. Vorlesung: $PSPACE$ -Vollständigkeit

Markus Krötzsch

Professur Wissensbasierte Systeme

TU Dresden, 4. Juni 2026

NL: Probleme, die eine nichtdeterministische TM mit logarithmischem Speicher lösen kann

- Deterministisch lösbar in polynomieller Zeit
- In der Regel deutlich besser parallelisierbar als typische Probleme in P
- Aber: seriell nicht unbedingt schneller lösbar als andere Probleme in P

**Typisches Beispiel: Erreichbarkeit** in gerichteten Graphen

# PSPACE

## PSPACE-vollständige Probleme

= Probleme, die mindestens so schwer sind, wie alle anderen Probleme in PSPACE

= die schwersten Probleme in PSPACE.

## **Vermutung:**

Kein PSPACE-vollständiges Problem ist in NP,

d.h. PSPACE ist echt schwerer als NP und coNP

## **PSPACE-vollständige Probleme**

- **TrueQBF:** Wahrheit von quantifizierten Booleschen Formeln (und das Formelspiel)
- **Geography:** Spiel auf einem Graphen

# $PSPACE$ -Vollständigkeit

## Rückblick: **TrueQBF** in $PSPACE$

**Satz:** **TrueQBF** ist in  $PSPACE$ .

## Rückblick: **TrueQBF** in $PSPACE$

**Satz:** TrueQBF ist in  $PSPACE$ .

**Beweis:** Durch Angabe eines (Pseudo-)Algorithmus

```
01 TRUEQBF( $F$ ) :  
02   if  $F$  „hat keine Quantoren“ :  
03     return „Aussagenlogische Auswertung von  $F$ “  
04   else if  $F = \exists p.G$  :  
05     return (TRUEQBF( $G[p/\top]$ ) OR TRUEQBF( $G[p/\perp]$ ))  
06   else if  $F = \forall p.G$  :  
07     return (TRUEQBF( $G[p/\top]$ ) AND TRUEQBF( $G[p/\perp]$ ))
```

- Evaluation in Zeile 03 möglich in  $PSPACE$
- Rekursion in Zeilen 05 und 07 können der Reihe nach abgearbeitet werden, wobei Speicher wiederverwendet wird
- Jeder Rekursionsschritt benötigt polynomiellen Speicher
- Maximale Rekursionstiefe = Zahl der Atome (linear) □

# PSPACE-Schwere

Ein Problem  $Q$  ist **PSPACE-schwer** wenn für jedes Problem  $P$  in PSPACE ein polynomielle Reduktion  $P \leq_p Q$  existiert.  $Q$  ist **PSPACE-vollständig** wenn es PSPACE-schwer ist und in PSPACE liegt.

# PSPACE-Schwere

Ein Problem  $Q$  ist **PSPACE-schwer** wenn für jedes Problem  $P$  in PSPACE ein polynomielle Reduktion  $P \leq_p Q$  existiert.  $Q$  ist **PSPACE-vollständig** wenn es PSPACE-schwer ist und in PSPACE liegt.

**Satz: TrueQBF** ist PSPACE-schwer.

# PSPACE-Schwere

Ein Problem  $Q$  ist **PSPACE-schwer** wenn für jedes Problem  $P$  in PSPACE ein polynomielle Reduktion  $P \leq_p Q$  existiert.  $Q$  ist **PSPACE-vollständig** wenn es PSPACE-schwer ist und in PSPACE liegt.

**Satz: TrueQBF** ist PSPACE-schwer.

## Beweisidee:

- Wie beim Satz von Cook/Levin stellen wir den Lauf einer Turingmaschine mit aussagenlogischen Atomen dar
- Speicher ist wie bei NP polynomiell (einfach kodierbar)
- Zeit ist problematisch: in PSPACE kann ein TM exponentiell lang laufen  $\leadsto$  wir können nicht einfach für jeden Zeitschritt eine neue Version der Konfigurationsvariablen anlegen und jeden Übergang mit Formeln axiomatisieren

# Beweisidee (Fortsetzung)

## Einsichten:

- Man kann eine komplette TM-Konfiguration in polynomiell vielen Atomen kodieren (wie bei Cook-Levin)
- Ebenso kann man zwei oder drei Konfigurationen kodieren – aber nicht exponentiell viele (für jeden Schritt)
- Direkte Übergänge, Start- und Endkonfiguration kann man mit Formeln axiomatisieren

# Beweisidee (Fortsetzung)

## **Einsichten:**

- Man kann eine komplette TM-Konfiguration in polynomiell vielen Atomen kodieren (wie bei Cook-Levin)
- Ebenso kann man zwei oder drei Konfigurationen kodieren – aber nicht exponentiell viele (für jeden Schritt)
- Direkte Übergänge, Start- und Endkonfiguration kann man mit Formeln axiomatisieren

## **Akzeptanz ausdrücken**

- Gewünschte Aussage: „Ausgehend von der Startkonfiguration kann eine Endkonfiguration in endlich vielen Übergängen erreicht werden“
- = Erreichbarkeitsproblem in einem exponentiell großen Graphen

# Beweisidee (Fortsetzung)

Erreichbarkeitsproblem in einem exponentiell großen Graphen

# Beweisidee (Fortsetzung)

Erreichbarkeitsproblem in einem exponentiell großen Graphen

**Lösung:** „Middle-First-Suche“

- Um zu prüfen, ob man von  $s$  nach  $t$  gelangen kann
- Prüfe zunächst, ob  $s = t$  oder  $s$  direkter Vorgänger von  $t$
- Rate einen Punkt  $m$  in der „Mitte“ eines Pfades von  $s$  nach  $t$
- Prüfe (rekursiv) ob man von  $s$  nach  $m$  gelangen kann
- Prüfe (rekursiv) ob man von  $m$  nach  $t$  gelangen kann

# Beweisidee (Fortsetzung)

## Erreichbarkeitsproblem in einem exponentiell großen Graphen

### **Lösung:** „Middle-First-Suche“

- Um zu prüfen, ob man von  $s$  nach  $t$  gelangen kann
- Prüfe zunächst, ob  $s = t$  oder  $s$  direkter Vorgänger von  $t$
- Rate einen Punkt  $m$  in der „Mitte“ eines Pfades von  $s$  nach  $t$
- Prüfe (rekursiv) ob man von  $s$  nach  $m$  gelangen kann
- Prüfe (rekursiv) ob man von  $m$  nach  $t$  gelangen kann

↪ Anzahl der zu ratenden Mittelpunkte ist logarithmisch in Länge des Pfades

↪ Speicher kann in jedem Schritt wiederverwendet werden

Diesen Algorithmus kann man polynomiell in QBF kodieren

# Alternierende Quantoren

**TrueQBF<sub>alt</sub>** ist das Problem **TrueQBF**, beschränkt auf QBF der Form  $\exists p_1. \forall p_2. \exists p_3. \dots \forall p_{\ell-1}. \exists p_{\ell}. F$ , wobei  $F$  in Klauselform ist.

Diese Version entspricht eher der Idee eines Spiels, bei dem Emilia und Anton abwechselnd Belegungen festlegen. Emilia erhält den ersten und den letzten Zug.

# Alternierende Quantoren

**TrueQBF<sub>alt</sub>** ist das Problem **TrueQBF**, beschränkt auf QBF der Form  $\exists p_1. \forall p_2. \exists p_3. \dots \forall p_{\ell-1}. \exists p_{\ell}. F$ , wobei  $F$  in Klauselform ist.

Diese Version entspricht eher der Idee eines Spiels, bei dem Emilia und Anton abwechselnd Belegungen festlegen. Emilia erhält den ersten und den letzten Zug.

**Satz:** **TrueQBF<sub>alt</sub>** ist PSPACE-vollständig.

# Alternierende Quantoren

**TrueQBF<sub>alt</sub>** ist das Problem **TrueQBF**, beschränkt auf QBF der Form  $\exists p_1. \forall p_2. \exists p_3. \dots \forall p_{\ell-1}. \exists p_{\ell}. F$ , wobei  $F$  in Klauselform ist.

Diese Version entspricht eher der Idee eines Spiels, bei dem Emilia und Anton abwechselnd Belegungen festlegen. Emilia erhält den ersten und den letzten Zug.

**Satz:** **TrueQBF<sub>alt</sub>** ist PSPACE-vollständig.

**Beweis:** **TrueQBF<sub>alt</sub>**  $\in$  PSPACE folgt aus **TrueQBF**  $\in$  PSPACE.

Für die Schwere zeigen wir **TrueQBF**  $\leq_p$  **TrueQBF<sub>alt</sub>**:

- man fügt einfach nach Bedarf zusätzliche Quantoren ein
- es müssen nicht alle quantifizierten Atome in  $F$  vorkommen □

# Rückblick: Geography

## Ein Kinderspiel:

- Zwei Spieler benennen abwechselnd Städte
- Jede Stadt muss mit dem letzten Buchstaben der zuvor genannten beginnen
- Wiederholungen sind verboten
- Der erste Spieler, der keine Stadt mehr nennen kann, verliert

## Ein Mathematikerspiel:

- Zwei Spieler markieren Knoten in einem gerichteten Graph
- Jeder Knoten muss ein Nachfolger des vorigen sein
- Wiederholungen sind verboten
- Der erste Spieler, der keinen Knoten markieren kann, verliert

Entscheidungsproblem **Geography**:

**Gegeben:** Ein gerichteter Graph und ein Startknoten

**Frage:** Hat Emilia eine Gewinnstrategie für dieses Spiel?

# Geography ist PSPACE-vollständig (1)

**Satz:** Geography ist PSPACE-vollständig.

# Geography ist PSPACE-vollständig (1)

**Satz:** Geography ist PSPACE-vollständig.

**Beweis:** (1) Geography ist in PSPACE

# Geography ist PSPACE-vollständig (1)

**Satz:** Geography ist PSPACE-vollständig.

**Beweis:** (1) Geography ist in PSPACE

Bei Startknoten  $s$  und Graph  $G$  erhalten wir die Antwort durch Aufruf von  $\text{ECANWIN}(G, s, \{s\})$ , definiert wie folgt:

```
01 ECANWIN( $G, n, \text{Visited}$ ) :  
02   result = false  
03   for all Nachfolger*  $m$  von  $n$  in  $G$  mit  $m \notin \text{Visited}$  :  
04     result = result OR ACANNOTWIN( $G, m, \text{Visited} \cup \{m\}$ )  
05   return result  
  
06 ACANNOTWIN( $G, n, \text{Visited}$ ) :  
07   result = true  
08   for all Nachfolger*  $m$  von  $n$  in  $G$  mit  $m \notin \text{Visited}$  :  
09     result = result AND ECANWIN( $G, m, \text{Visited} \cup \{m\}$ )  
10   return result
```

\* gemeint sind jeweils direkte Nachfolger

# Geography ist PSPACE-vollständig (2)

**Satz:** Geography ist PSPACE-vollständig.

# Geography ist PSPACE-vollständig (2)

**Satz:** Geography ist PSPACE-vollständig.

**Beweis:** (2) Geography ist PSPACE-schwer

# Geography ist PSPACE-vollständig (2)

**Satz:** Geography ist PSPACE-vollständig.

**Beweis:** (2) Geography ist PSPACE-schwer

Durch Reduktion  $\text{TrueQBF}_{\text{alt}} \leq_p \text{Geography}$ :

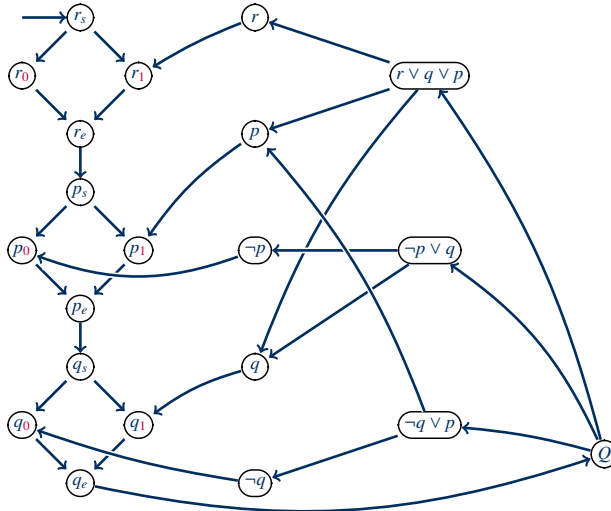
Wir betrachten eine QBF  $\exists p_1. \forall p_2 \dots \exists p_\ell. F$  und konstruieren daraus einen Graphen für Geography

**Grundidee:** (siehe auch M. Sipser: Introduction to the Theory of Computation; 2013; Thm. 8.14)

- Der Graph beginnt mit einer Abfolge von Rauten-Strukturen, welche die Entscheidungen der Spieler darstellen: für jedes Atom  $p$  gibt es dort zwei Knoten,  $p_1$  und  $p_0$ , von denen genau einer besucht wird
- Es folgt eine Baumstruktur: ein Knoten für  $F$ , darunter ein Knoten für jede Klausel und darunter jeweils ein Knoten für jedes Literal
- Von jedem Literal  $p$  bzw.  $\neg p$  gibt es eine Kante zurück zum Knoten  $p_1$  bzw.  $p_0$

# Geography ist PSPACE-vollständig: Beispiel

Wir betrachten die Formel  $\exists r. \forall p. \exists q. (r \vee q \vee p) \wedge (\neg p \vee q) \wedge (\neg q \vee p)$



# Geography ist $PSPACE$ -vollständig (3)

**Satz:** Geography ist  $PSPACE$ -vollständig.

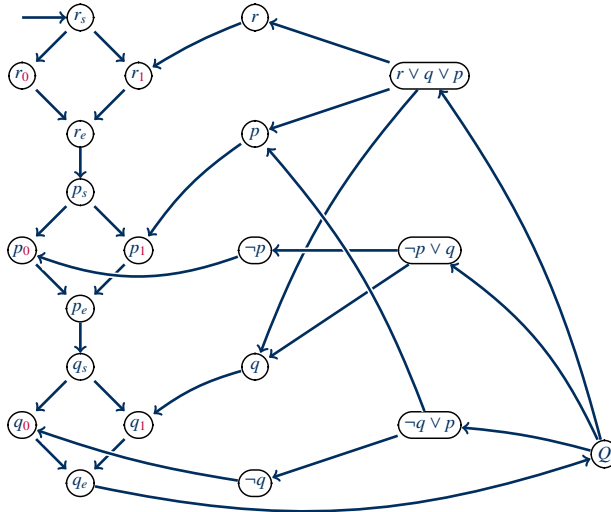
**Beweis:** (2) Geography ist  $PSPACE$ -schwer

Beweisidee:

- Zuerst bestimmen Anton und Emilia abwechselnd Wahrheitswerte, indem sie die Rauten jeweils links oder rechts durchlaufen
- Danach darf Anton zur Auswertung eine Klausel auswählen (die er für nicht erfüllt hält)
- Anschließend wählt Emilia ein Literal dieser Klausel (welches ihrer Meinung nach erfüllt ist)
- Emilia gewinnt, wenn das Literal wirklich erfüllt war (denn dann gibt es keinen Weg zu einem noch nicht besuchten Knoten)
- Anton gewinnt, wenn das Literal nicht erfüllt war (denn dann kann er den Weg noch genau einen Schritt fortsetzen)

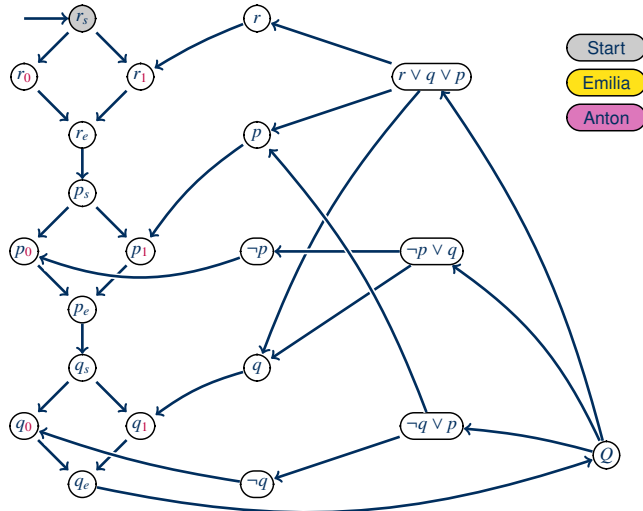
# Geography ist PSPACE-vollständig: Beispiel

Wir betrachten die Formel  $\exists r. \forall p. \exists q. (r \vee q \vee p) \wedge (\neg p \vee q) \wedge (\neg q \vee p)$



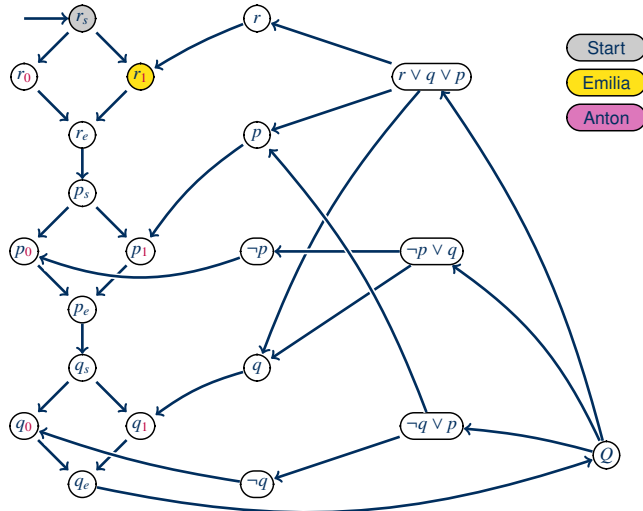
# Geography ist PSPACE-vollständig: Beispiel

Wir betrachten die Formel  $\exists r. \forall p. \exists q. (r \vee q \vee p) \wedge (\neg p \vee q) \wedge (\neg q \vee p)$



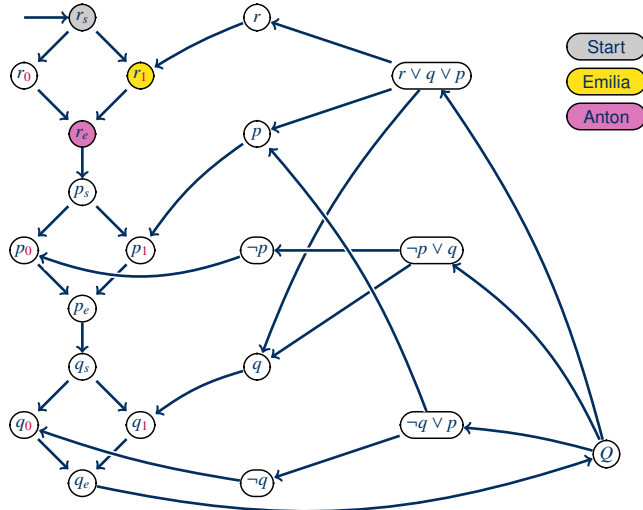
# Geography ist PSPACE-vollständig: Beispiel

Wir betrachten die Formel  $\exists r. \forall p. \exists q. (r \vee q \vee p) \wedge (\neg p \vee q) \wedge (\neg q \vee p)$



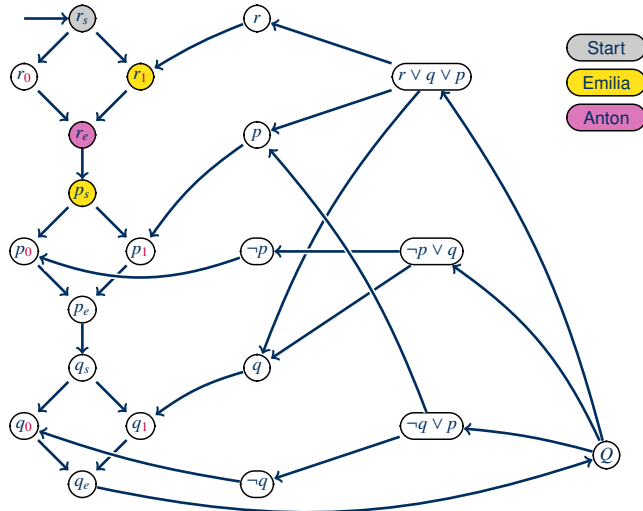
# Geography ist PSPACE-vollständig: Beispiel

Wir betrachten die Formel  $\exists r. \forall p. \exists q. (r \vee q \vee p) \wedge (\neg p \vee q) \wedge (\neg q \vee p)$



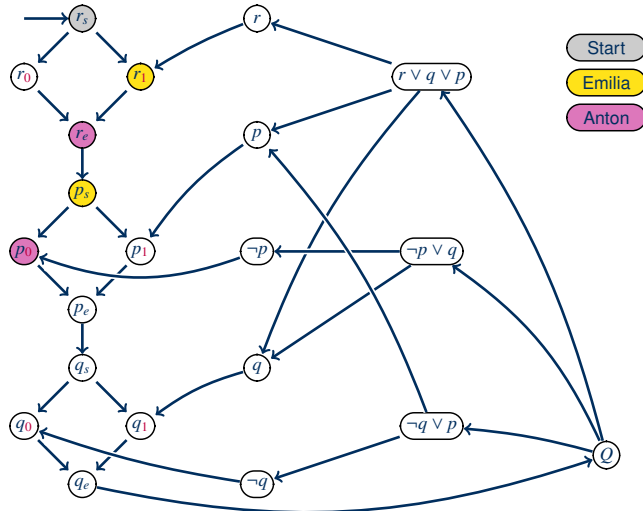
# Geography ist PSPACE-vollständig: Beispiel

Wir betrachten die Formel  $\exists r. \forall p. \exists q. (r \vee q \vee p) \wedge (\neg p \vee q) \wedge (\neg q \vee p)$



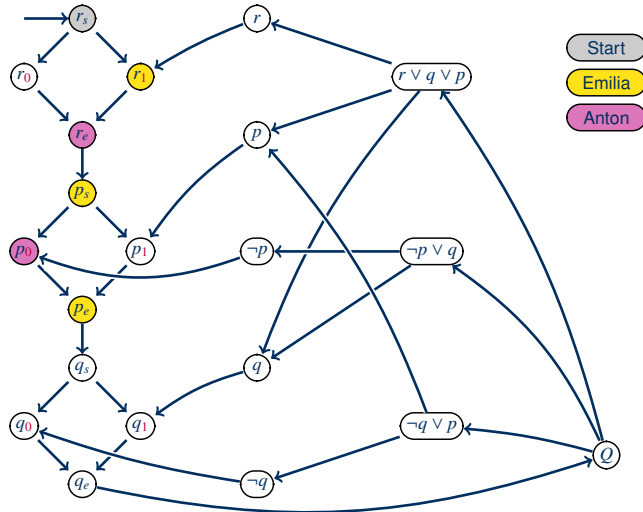
# Geography ist PSPACE-vollständig: Beispiel

Wir betrachten die Formel  $\exists r. \forall p. \exists q. (r \vee q \vee p) \wedge (\neg p \vee q) \wedge (\neg q \vee p)$



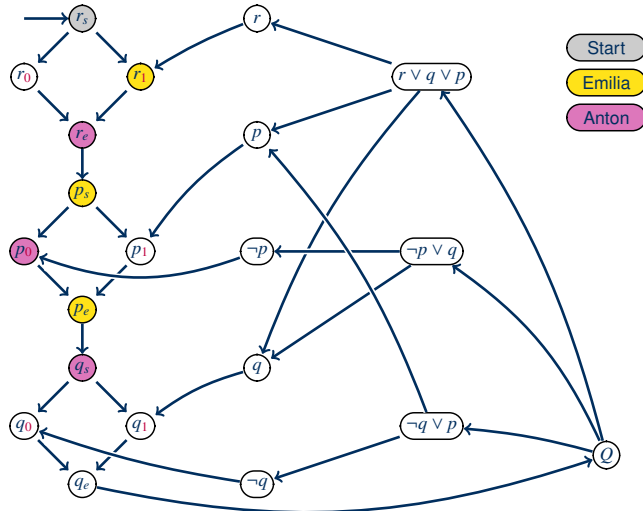
# Geography ist PSPACE-vollständig: Beispiel

Wir betrachten die Formel  $\exists r. \forall p. \exists q. (r \vee q \vee p) \wedge (\neg p \vee q) \wedge (\neg q \vee p)$



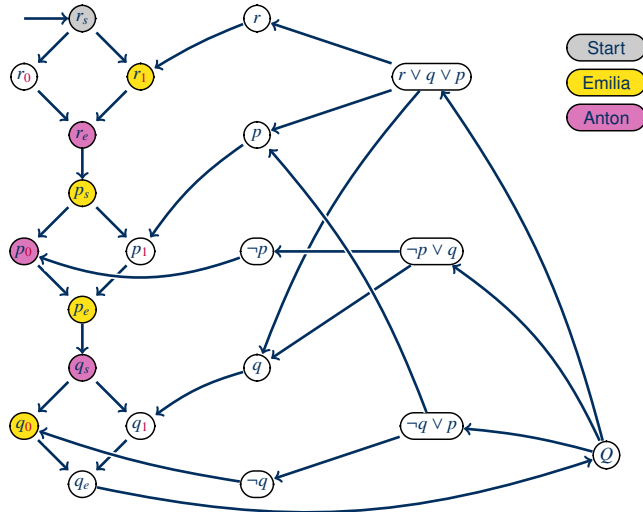
# Geography ist PSPACE-vollständig: Beispiel

Wir betrachten die Formel  $\exists r. \forall p. \exists q. (r \vee q \vee p) \wedge (\neg p \vee q) \wedge (\neg q \vee p)$



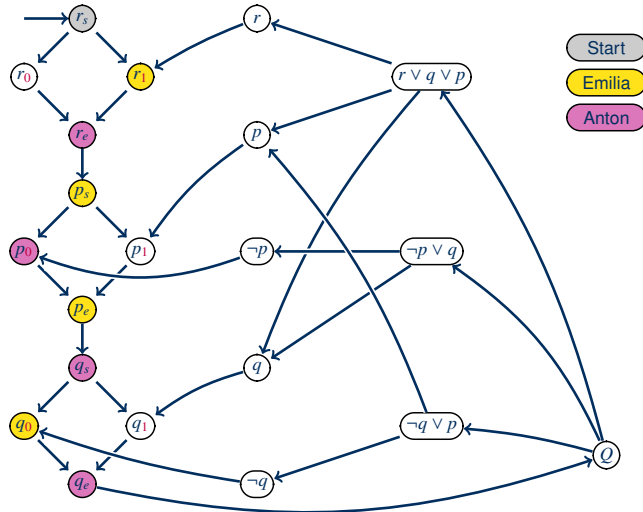
# Geography ist PSPACE-vollständig: Beispiel

Wir betrachten die Formel  $\exists r. \forall p. \exists q. (r \vee q \vee p) \wedge (\neg p \vee q) \wedge (\neg q \vee p)$



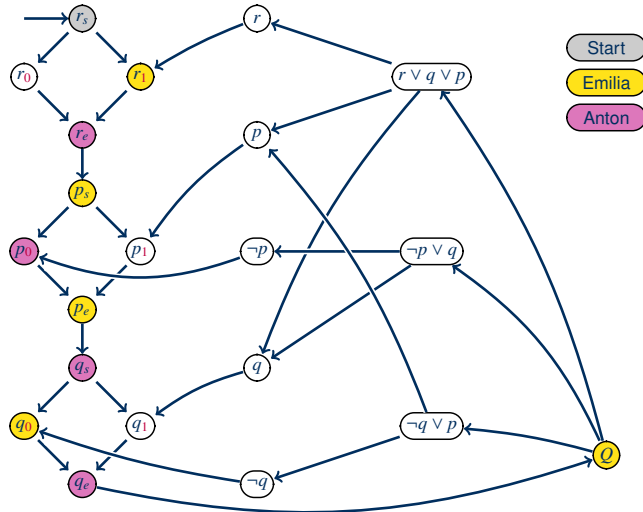
# Geography ist PSPACE-vollständig: Beispiel

Wir betrachten die Formel  $\exists r. \forall p. \exists q. (r \vee q \vee p) \wedge (\neg p \vee q) \wedge (\neg q \vee p)$



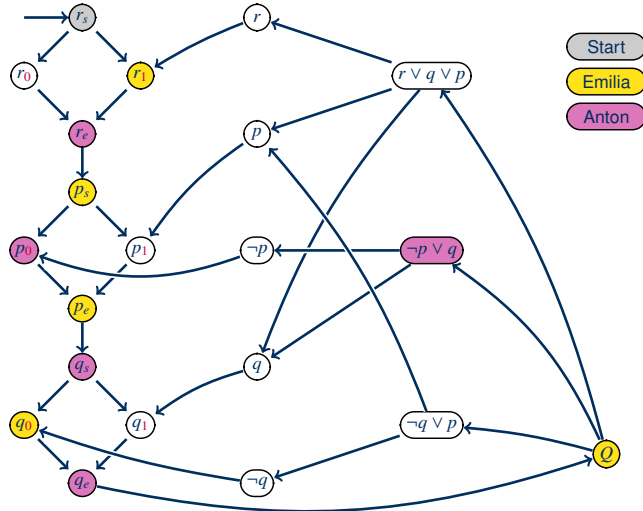
# Geography ist PSPACE-vollständig: Beispiel

Wir betrachten die Formel  $\exists r. \forall p. \exists q. (r \vee q \vee p) \wedge (\neg p \vee q) \wedge (\neg q \vee p)$



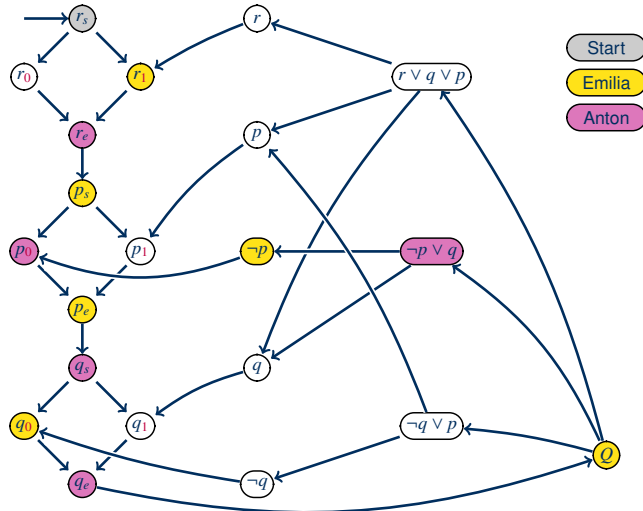
# Geography ist PSPACE-vollständig: Beispiel

Wir betrachten die Formel  $\exists r. \forall p. \exists q. (r \vee q \vee p) \wedge (\neg p \vee q) \wedge (\neg q \vee p)$



# Geography ist PSPACE-vollständig: Beispiel

Wir betrachten die Formel  $\exists r. \forall p. \exists q. (r \vee q \vee p) \wedge (\neg p \vee q) \wedge (\neg q \vee p)$



# Weitere Komplexitätsklassen

# Komplexität jenseits von PSPACE?

Wir haben uns auf drei Klassen konzentriert:

$$P \subseteq NP \subseteq PSPACE$$

Ersetzt man „polynomiell“ durch „exponentiell“, so erhält man jenseits von PSPACE ein ganz ähnliches Bild:

$$EXPTIME \subseteq NEXPTIME \subseteq EXPSPACE$$

# Komplexität jenseits von PSPACE?

Wir haben uns auf drei Klassen konzentriert:

$$P \subseteq NP \subseteq PSPACE$$

Ersetzt man „polynomiell“ durch „exponentiell“, so erhält man jenseits von PSPACE ein ganz ähnliches Bild:

$$EXPTIME \subseteq NEXPTIME \subseteq EXPSPACE$$

Man kann beliebig hohe Türme von Exponenten konstruieren:

$$2EXPTIME \subseteq N2EXPTIME \subseteq 2EXPSPACE$$

$$3EXPTIME \subseteq N3EXPTIME \subseteq 3EXPSPACE$$

⋮

Vieles, was wir gelernt haben, gilt hier wie zuvor – aber diese Klassen sind von immer geringerer praktischer Relevanz

# Gibt es so schwere Probleme?

## Gibt es so schwere Probleme?

Für  $k$ -Exp-Klassen lassen sich künstliche Probleme leicht finden: „Gegeben eine det. Turingmaschine  $\mathcal{M}$ , eine Eingabe  $w$ , und ein ‘Füllwort’  $a^m$  der Länge  $m$ , wird  $\mathcal{M}$  das Wort  $w$  in Zeit  $\underbrace{2^{\cdot 2^m}}_{k\text{-mal } 2}$  akzeptieren?“ (Warum ist das  $k$ -Exp-vollständig? Zeigen Sie Inklusion und Schwere.)

## Gibt es so schwere Probleme?

Für  $k$ -EXP-Klassen lassen sich künstliche Probleme leicht finden: „Gegeben eine det. Turingmaschine  $\mathcal{M}$ , eine Eingabe  $w$ , und ein ‘Füllwort’  $a^m$  der Länge  $m$ , wird  $\mathcal{M}$  das Wort  $w$  in Zeit  $\underbrace{2^{\cdot^{2^m}}}_{k\text{-mal } 2}$  akzeptieren?“ (Warum ist das  $k$ -Exp-vollständig? Zeigen Sie Inklusion und Schwere.)

Jenseits von EXPSPACE gibt es nur wenige relevante Probleme.

**Beispiel:** Der W3C-Standard [Web Ontology Language](#) (OWL, Version 2) definiert die logische Beschreibungssprache OWL 2 DL. Logisches Schließen in dieser Sprache ist N2EXPTIME-vollständig.

## Gibt es so schwere Probleme?

Für  $k$ -EXP-Klassen lassen sich künstliche Probleme leicht finden: „Gegeben eine det. Turingmaschine  $\mathcal{M}$ , eine Eingabe  $w$ , und ein ‘Füllwort’  $a^m$  der Länge  $m$ , wird  $\mathcal{M}$  das Wort  $w$  in Zeit  $\underbrace{2^{\cdot^{2^m}}}_{k\text{-mal } 2}$  akzeptieren?“ (Warum ist das  $k$ -Exp-vollständig? Zeigen Sie Inklusion und Schwere.)

Jenseits von  $\text{EXPSPACE}$  gibt es nur wenige relevante Probleme.

**Beispiel:** Der W3C-Standard [Web Ontology Language](#) (OWL, Version 2) definiert die logische Beschreibungssprache OWL 2 DL. Logisches Schließen in dieser Sprache ist  $\text{N2EXP TIME}$ -vollständig.

Es gibt aber auch entscheidbare Probleme, die durch keine vielfach exponentiell beschränkte TM entschieden werden: Dies sind Probleme **nicht-elementarer** Komplexität

**Beispiel:** Die Äquivalenz von regulären Ausdrücken mit einem zusätzlichen Komplexifizierungsoperator ist entscheidbar, aber nicht elementar.

## Gibt es so schwere Probleme?

Für  $k$ -Exp-Klassen lassen sich künstliche Probleme leicht finden: „Gegeben eine det. Turingmaschine  $\mathcal{M}$ , eine Eingabe  $w$ , und ein ‘Füllwort’  $a^m$  der Länge  $m$ , wird  $\mathcal{M}$  das Wort  $w$  in Zeit  $\underbrace{2^{\cdot^{2^m}}}_{k\text{-mal } 2}$  akzeptieren?“ (Warum ist das  $k$ -Exp-vollständig? Zeigen Sie Inklusion und Schwere.)

Jenseits von  $\text{ExpSPACE}$  gibt es nur wenige relevante Probleme.

**Beispiel:** Der W3C-Standard [Web Ontology Language](#) (OWL, Version 2) definiert die logische Beschreibungssprache OWL 2 DL. Logisches Schließen in dieser Sprache ist  $\text{N2ExpTIME}$ -vollständig.

Es gibt aber auch entscheidbare Probleme, die durch keine vielfach exponentiell beschränkte TM entschieden werden: Dies sind Probleme **nicht-elementarer** Komplexität

**Beispiel:** Die Äquivalenz von regulären Ausdrücken mit einem zusätzlichen Komplementierungsoperator ist entscheidbar, aber nicht elementar.

Viele schwere praktische Probleme sind einfach unentscheidbar

(z.B. Syntax von C++)

# Komplexitäten unterhalb von P?

Wir haben die folgenden Komplexitäten betrachtet

$$L \subseteq NL \subseteq P$$

Hier gibt es bereits **technische Probleme:**

- TMs mit logarithmischem Speicher benötigen getrennte Ein- und Ausgabebänder
- Logarithmen sind nicht abgeschlossen unter polynomiellen Operationen; z.B. sagt uns Savitch nicht ob  $L \stackrel{?}{=} NL$

# Komplexitäten unterhalb von P?

Wir haben die folgenden Komplexitäten betrachtet

$$L \subseteq NL \subseteq P$$

Hier gibt es bereits **technische Probleme:**

- TMs mit logarithmischem Speicher benötigen getrennte Ein- und Ausgabebänder
- Logarithmen sind nicht abgeschlossen unter polynomiellen Operationen; z.B. sagt uns Savitch **nicht** ob  $L \stackrel{?}{=} NL$

Will man **noch kleinere Komplexitätsklassen** betrachten, dann muss man zu anderen Berechnungsmodellen greifen:

- Schaltkreise statt TMs ( $\leadsto$  **circuit complexity**)
- Logarithmisch zeitbeschränkte TMs mit beschränkter Parallelverarbeitung ( $\leadsto$  **alternating, random access TM**)
- Automatenmodelle ( $\leadsto$  DFA, PDA, ...)

Manche der Klassen sind dann sehr stark von Details der Problemdefinition abhängig (wenig robust)

# Gibt es so leichte Probleme?

Auch bei den subpolynomiellen Klassen ergeben sich in der Regel typische Probleme aus der Definition.

# Gibt es so leichte Probleme?

Auch bei den subpolynomiellen Klassen ergeben sich in der Regel typische Probleme aus der Definition.

Es gibt eine Reihe **interessanter, praktisch relevanter L-Probleme**, z.B.

- Erreichbarkeit in ungerichteten Graphen (Omar Reingold, 2005)
- Zwei-Färbbarkeit von Graphen

# Gibt es so leichte Probleme?

Auch bei den subpolynomiellen Klassen ergeben sich in der Regel typische Probleme aus der Definition.

Es gibt eine Reihe **interessanter, praktisch relevanter L-Probleme**, z.B.

- Erreichbarkeit in ungerichteten Graphen (Omar Reingold, 2005)
- Zwei-Färbbarkeit von Graphen

Es gibt auch **noch einfachere praktisch relevante Probleme**, z.B.

- logische und arithmetische Operationen
- Beantwortung von fest vorgegebenen SQL-Anfragen auf beliebigen Datenbanken
- Wortprobleme bei endlichen Automaten

In diesen Fällen wird es immer schwerer, von „Schwere“ und „Vollständigkeit“ zu sprechen

# Komplexität und Spiele

NP ist eine typische Klasse für Solitaire-Spiele:

- Sudoku, Minesweeper, Tetris, ...

# Komplexität und Spiele

NP ist eine typische Klasse für Solitaire-Spiele:

- Sudoku, Minesweeper, Tetris, ...

PSPACE ist eine typische Klasse für Spiele, bei denen zwei Spieler abwechselnd eine polynomielle Zahl an Zügen durchführen:

- Geography, Reversi, Tic-Tac-Toe, aber auch: Sokoban, ...

# Komplexität und Spiele

NP ist eine typische Klasse für Solitaire-Spiele:

- Sudoku, Minesweeper, Tetris, ...

PSPACE ist eine typische Klasse für Spiele, bei denen zwei Spieler abwechselnd eine polynomielle Zahl an Zügen durchführen:

- Geography, Reversi, Tic-Tac-Toe, aber auch: Sokoban, ...

EXPTIME findet sich bei Spielen, bei denen man Züge rückgängig machen kann (polynomielles Spielbrett – exponentiell viele Züge):

- Schach, Dame, Go, Stern-Halma, ...

In jedem Fall muss man (nicht-endliche) Verallgemeinerungen der Spiele betrachten, um die „wahre“ Komplexität zu sehen

(Menschen spielen nicht, indem sie innerlich eine endliche Datenbank aller möglichen Stellungen konsultieren)

# Komplexität und Spiele

NP ist eine typische Klasse für Solitaire-Spiele:

- Sudoku, Minesweeper, Tetris, ...

PSPACE ist eine typische Klasse für Spiele, bei denen zwei Spieler abwechselnd eine polynomielle Zahl an Zügen durchführen:

- Geography, Reversi, Tic-Tac-Toe, aber auch: Sokoban, ...

EXPTIME findet sich bei Spielen, bei denen man Züge rückgängig machen kann (polynomielles Spielbrett – exponentiell viele Züge):

- Schach, Dame, Go, Stern-Halma, ...

In jedem Fall muss man (nicht-endliche) Verallgemeinerungen der Spiele betrachten, um die „wahre“ Komplexität zu sehen

(Menschen spielen nicht, indem sie innerlich eine endliche Datenbank aller möglichen Stellungen konsultieren)

**Spiele sollten komplex sein, um lange zu motivieren**

# Und sonst so?

Es gibt viele weitere Themen in der Komplexitätstheorie

↪ siehe Vorlesung „Complexity Theory“ (Wintersemester)

# Zusammenfassung und Ausblick

**TrueQBF** und **Geography** sind  $PSPACE$ -vollständig

Wir kennen die praktisch wichtigsten Komplexitätsklassen und jeweils typische Probleme

Spiele liefern interessante Beispiele komplexer Probleme

Was erwartet uns als nächstes?

- Einführung in die Prädikatenlogik
- Entscheidbare logische Probleme
- Unentscheidbare logische Probleme