Vollständigkeit in Komplexitätsklassen unterhalb von P:

Für Komplexitätsklassen $\mathscr{K} \subseteq$ P liefern *Polynomialzeit*-Reduktionen keinen sinnvollen Vollständigkeitsbegriff, denn jedes Problem in $\mathscr{K}$ (außer den "trivialen" Problemen $\varnothing$ und Σ^*) ist vollständig (bzgl. Polynomialzeit-Reduktionen) für $\mathscr{K}$.
Um einen sinnvollen Vollständigkeitsbegriff für Klassen $\mathscr{K}$ mit LOGSPACE $\subset \mathscr{K} \subseteq$ P zu erhalten, werden oft die *Logspace*-Reduktionen verwendet, die analog zu den Polynomialzeit-Reduktionen definiert sind, wobei die Funktion f auf logarithmischem Platz berechenbar sein musss. Ein Beispiel für ein solchermaßen NLOGSPACE-vollständiges Problem ist das bereits in Kapitel 0 (Einleitung) betrachtete Problem ERREICHBARKEIT.

Eine andere Charakterisierung der Klasse NP:

Oft werden nichtdeterministische Polynomialzeit-Algorithmen so beschrieben, dass der Algorithmus eine Lösung "rät" und diese dann überprüft. Beispielsweise kann ein Algorithmus, der TSP löst, eine Rundreise erraten und dann leicht ausrechnen, ob deren Gesamtlänge auch wirklich $\leqslant k$ ist.
Die folgende Charakterisierung von NP macht diesen Ansatz explizit:

1.39 Satz. *Ein Problem $L \subseteq \Sigma^*$ ist genau dann in* NP*, wenn es ein Polynom $p(n)$ und ein Problem $L' \in$ P gibt, so dass*

$$L \;=\; \{w \in \Sigma^* \mid \textit{es gibt ein Wort } z\textit{, so dass } |z| \leqslant p(|w|) \textit{ und } w\#z \in L'\}.$$

Das Wort z wird oft *polynomieller Zeuge* genannt.

Beweis: Ein direkter Beweis über Turing-Maschinen ist sehr leicht.
Der Satz folgt aber auch aus dem *Satz von Fagin*, der in Kapitel 2 (Deskriptive Komplexität) bewiesen wird. □

1.2.4 Orakel-Turing-Maschinen und die Polynomialzeit-Hierarchie

Die Polynomialzeit-Hierarchie ist wegen ihres engen Zusammenhangs zur Logik zweiter Stufe für diese Vorlesung von Interesse; siehe Kapitel 2 (Deskriptive Komplexität). Um die Komplexitätsklassen der Polynomialzeit-Hierarchie einführen zu können, benötigen wir den Begriff der *Orakel-Turing-Maschinen*.

1.40 Definition (Orakel-TM).
Sei Σ ein Alphabet und $B \subseteq \Sigma^*$ eine Sprache.
Eine *Orakel-Turing-Maschine M mit Orakel B* ist eine Turing-Maschine mit einem zusätzlichen Band, dem so genannten *Orakel-Band* und 3 zusätzlichen Zuständen q_{ja}, q_{nein} und $q_?$.
Berechnungen und Konfigurationen von M sind wie üblich definiert, nur hat die Orakel-TM zusätzlich die Möglichkeit, Anfragen an das Orakel zu stellen. Dazu kann sie ein Wort w auf das Orakel-Band schreiben und dann in den Zustand $q_?$ gehen. Falls $w \in B$ liegt, so geht die Maschine direkt in den Zustand q_{ja} über; andernfalls in q_{nein}. In jedem der beiden Fälle wird der Inhalt des Orakel-Bands gelöscht.

Die Sprache B dient hier also als "Orakel". Die Entscheidung, ob $w \in B$ ist, ist "atomar", d.h. sie benötigt nur *einen* Schritt. Dies kann hilfreich sein, falls die Komplexität von B höher ist als die Maschine M selbst berechnen kann, z.B. wenn M deterministisch und polynomiell zeitbeschränkt ist und $B \in \mathrm{NP}$.

Das Befragen eines "Orakels" kann man sich als Anfragen an einen Netzwerk-Server vorstellen: Das Programm auf einem Netzwerk-Client darf wegen der geringen Rechenkapazität des Clients nur polynomiell lange laufen, wohingegen der Server komplexere Aufgaben bewältigen kann. Der Client kann daher in seiner Berechnung Anfragen an den Server stellen, den Server also als "Orakel" benutzen.

Orakel-Turing-Maschinen können benutzt werden, um neue Komplexitätsklassen einzuführen:

1.41 Definition. Sei $B \subseteq \Sigma^*$.

(a) Sei M eine Orakel-TM mit Orakel B. Die von M mit Orakel B akzeptierte Sprache ist

$$L(M^B) \;:=\; \{w \mid M \text{ akzeptiert Eingabe } w \text{ mit Orakel } B\}.$$

Des Weiteren werden folgende Komplexitätsklassen definiert:

(b) $$\mathrm{P}^B \;:=\; \left\{ L \;\middle|\; \begin{array}{l} \text{es gibt eine polynomiell zeitbeschränkte deterministische} \\ \text{Orakel-TM } M \text{ mit Orakel } B\text{, die } L \text{ entscheidet} \end{array} \right\}.$$

(b) $$\mathrm{NP}^B \;:=\; \left\{ L \;\middle|\; \begin{array}{l} \text{es gibt eine polynomiell zeitbeschränkte nichtdeterministische} \\ \text{Orakel-TM } M \text{ mit Orakel } B\text{, die } L \text{ entscheidet} \end{array} \right\}.$$

(c) Für eine Komplexitätsklasse $\mathscr{K}$ ist

$$\mathrm{P}^{\mathscr{K}} \;:=\; \bigcup_{B \in \mathscr{K}} \mathrm{P}^B \qquad \text{und} \qquad \mathrm{NP}^{\mathscr{K}} \;:=\; \bigcup_{B \in \mathscr{K}} \mathrm{NP}^B.$$

1.42 Bemerkung. Offensichtlich gilt:

- $\mathrm{P}^{\mathrm{P}} = \mathrm{P}$, denn anstatt das Orakel zu befragen kann eine TM die Antwort des Orakels auch selbst berechnen.
- $\mathrm{NP} \subseteq \mathrm{P}^{\mathrm{NP}}$ und $\text{co-NP} \subseteq \mathrm{P}^{\mathrm{NP}}$.
- $\mathrm{P}^{\mathrm{NP}} = \mathrm{P}^B$ für jedes NP-vollständige Problem B.
  Insbesondere gilt also $\mathrm{P}^{\mathrm{NP}} = \mathrm{P}^{\textsc{Sat}} = \mathrm{P}^{\textsc{Tsp}} = \mathrm{P}^{\textsc{Clique}}$.
- $\mathrm{P}^{\mathrm{NP}} \subseteq \textsc{Pspace}$.

Erweitert man P um NP-Orakel, so erhält man vermutlich echt mehr als NP und co-NP. Dies liegt daran, dass nichtdeterministische Zeitklassen wie NP vermutlich nicht unter Komplementbildung abgeschlossen sind (wegen der Asymmetrie in der Definition des Akzeptierens nichtdeterministischer Turing-Maschinen).

Sprachen, die durch Orakel-Maschinen definiert sind, können natürlich selbst wieder als Orakel verwendet werden. Dies führt zum Begriff der *Polynomialzeit-Hierarchie* (auch: *Polynomielle Hierarchie* bzw. *Polynomiale Hierarchie*):

1.43 Definition (Polynomialzeit-Hierarchie). Für jedes $k \in \mathbb{N}$ definieren wir induktiv die Komplexitätsklassen Σ_k^p, Π_k^p und Δ_k^p folgendermaßen:

- $\Delta_0^p \;:=\; \Sigma_0^p \;:=\; \Pi_0^p \;:=\; \mathrm{P}.$
- $\Delta_{k+1}^p \;:=\; \mathrm{P}^{\Sigma_k^p}, \qquad \Sigma_{k+1}^p \;:=\; \mathrm{NP}^{\Sigma_k^p}, \qquad \Pi_{k+1}^p \;:=\; \text{co-}\Sigma_{k+1}^p.$

Des Weiteren setzen wir $\mathrm{PH} \;:=\; \bigcup_{k\in\mathbb{N}} \Sigma_k^p.$

1.44 Bemerkung. Man sieht leicht, dass

$$\Delta_1^p \;=\; \mathrm{P}, \quad \Sigma_1^p \;=\; \mathrm{NP} \quad \text{und} \quad \Pi_1^p \;=\; \text{co-NP},$$

$$\Delta_k^p \;\subseteq\; \Sigma_k^p \;\subseteq\; \Delta_{k+1}^p \quad \text{und} \quad \Delta_k^p \;\subseteq\; \Pi_k^p \;\subseteq\; \Delta_{k+1}^p,$$

und

$$\mathrm{PH} \;\subseteq\; \mathrm{PSPACE}.$$

Insgesamt ergibt sich die in Abbildung 1.2 dargestellte Inklusionsstruktur der Klassen der Polynomiellen Hierarchie.

1.3 Die Komplexität der Logik erster Stufe und der Satz von Trachtenbrot

Wie in Kapitel 0 (Einleitung) dargestellt, sind die folgenden Logikprobleme von besonderem Interesse für die verschiedensten Bereiche der Informatik.

1.45 Definition (Logikprobleme). Sei $\mathscr{L}$ eine Logik und K eine Klasse von Strukturen.

(a) ERFÜLLBARKEITSPROBLEM FÜR $\mathscr{L}$ IN K:
Eingabe: Ein Satz $\varphi \in \mathscr{L}$.
Frage: Gibt es eine Struktur $\mathfrak{A} \in \mathsf{K}$ mit $\mathfrak{A} \models \varphi$?

(b) AUSWERTUNGSPROBLEM FÜR $\mathscr{L}$ UND K:
Eingabe: Ein Satz $\varphi \in \mathscr{L}$ und eine Struktur $\mathfrak{A} \in \mathsf{K}$.
Frage: Gilt $\mathfrak{A} \models \varphi$?

(c) BERECHNUNGSPROBLEM FÜR $\mathscr{L}$ UND K:
Eingabe: Eine Formel $\varphi(x_1, .., x_r) \in \mathscr{L}$ und eine Struktur $\mathfrak{A} \in \mathsf{K}$.
Ziel: Berechne $\varphi(\mathfrak{A}) := \{(a_1, .., a_r) \in A^r \mid \mathfrak{A} \models \varphi[a_1, .., a_r]\}$.

$$\begin{tikzcd}[row sep=small, column sep=small]
 & \textsc{Pspace} & \\
 & \mathrm{PH} \arrow[u, phantom, sloped, "\subseteq"] & \\
 & \vdots \arrow[u, phantom, sloped, "\subseteq"] & \\
 & \Delta^p_{k+1} \arrow[u, phantom, sloped, "\subseteq"] & \\
\Sigma^p_k \arrow[ur, phantom, sloped, "\subseteq"] & & \Pi^p_k \arrow[ul, phantom, sloped, "\supseteq"] \\
 & \Delta^p_3 \arrow[ul, phantom, sloped, "\supseteq"] \arrow[ur, phantom, sloped, "\subseteq"] & \\
\Sigma^p_2 \arrow[ur, phantom, sloped, "\subseteq"] & & \Pi^p_2 \arrow[ul, phantom, sloped, "\supseteq"] \\
 & \Delta^p_2 \arrow[ul, phantom, sloped, "\supseteq"] \arrow[ur, phantom, sloped, "\subseteq"] & \\
\mathrm{NP} = \Sigma^p_1 \arrow[ur, phantom, sloped, "\subseteq"] & & \Pi^p_1 = \text{co-NP} \arrow[ul, phantom, sloped, "\supseteq"] \\
 & \mathrm{P} = \Sigma^p_0 = \Delta^p_0 = \Pi^p_0 = \Delta^p_1 \arrow[ul, phantom, sloped, "\supseteq"] \arrow[ur, phantom, sloped, "\subseteq"] &
\end{tikzcd}$$

Abbildung 1.2: Inklusionsstruktur der Komplexitätsklassen der Polynomialzeit-Hierarchie.

(d) SUBSUMPTIONSPROBLEM FÜR $\mathscr{L}$ IN K (auch bekannt als *Query Containment Problem*):
Eingabe: Zwei Formeln $\varphi_1(x_1,..,x_r)$ und $\varphi_2(x_1,..,x_r) \in \mathscr{L}$.
Frage: Gilt für alle $\mathfrak{A} \in \mathsf{K}$, dass $\varphi_1(\mathfrak{A}) \subseteq \varphi_2(\mathfrak{A})$?

In diesem Abschnitt untersuchen wir die Komplexität der obigen Probleme für die *Logik erster Stufe* FO und die Klasse Fin aller *endlichen Strukturen*.

1.3.1 Die Standardkodierung von Strukturen und Formeln

Komplexitätsanalysen der Komplexitätstheorie basieren auf Wortsprachen, d.h. Klassen von Wörtern über einem endlichen Alphabet. Logikprobleme hingegen sind für Formeln einer Logik und Klassen von Strukturen definiert. Um über die Komplexität von Logikproblemen sprechen zu können, müssen wir deshalb Strukturen und Formeln als *Wörter* über einem geeigneten Alphabet kodieren.

Formeln lassen sich ganz einfach als Wörter kodieren — jede Formel φ der Logik erster Stufe lässt sich beispielsweise direkt als Wort *enc*(φ) über dem (endlichen) Alphabet

$$\{\, \wedge,\ \vee,\ \neg,\ \rightarrow,\ \leftrightarrow,\ (,\ ),\ \exists,\ \forall,\ =,\ \text{var},\ R,\ c,\ {\scriptstyle 0},\ {\scriptstyle 1},\ {\scriptstyle 2},\ {\scriptstyle 3},\ {\scriptstyle 4},\ {\scriptstyle 5},\ {\scriptstyle 6},\ {\scriptstyle 7},\ {\scriptstyle 8},\ {\scriptstyle 9}\,\}$$

auffassen.

Um auch *endliche Strukturen* durch Wörter über einem endlichen Alphabet repräsentieren zu können, benötigen wir zunächst ein paar grundlegende Begriffe.
Auf endlichen linearen Ordnungen kann man die Elemente der Ordnung eindeutig mit natürlichen Zahlen identifizieren, nämlich ihrem *Rang* (d.h. ihrer "Höhe") innerhalb der Ordnung:

1.46 Definition (Rang eines Elements bzgl. einer Ordnung). Sei A eine endliche, durch eine Relation $<$ linear geordnete Menge. Für ein Element $a \in A$ definieren wir den *Rang* von a bzgl. $<$ als

$$rg_<(a) \ := \ |\{b \in A \mid b < a\}|.$$

Falls $A = \{a_0 < a_1 < \cdots < a_{n-1}\}$, so ist $rg(a_i)$ also gerade die Zahl i.

1.47 Bemerkung. Alternativ lässt sich der Rang auch induktiv definieren als

$$rg_<(a) \ := \ \begin{cases} 0 & \text{falls es kein } b \in A \text{ mit } b < a \text{ gibt} \\ \max\{rg(b)+1 \mid b \in A \text{ mit } b < a\} & \text{sonst.} \end{cases}$$

Diese Definition ist auf *endlichen* Mengen A äquivalent zur obigen, funktioniert allerdings auch für bestimmte unendliche und partielle Ordnungen (so genannte *Wohlordnungen*).

1.48 Definition (Lexikographische Ordnung). Sei $A := \{a_0 < a_1 < \cdots < a_{n-1}\}$ eine durch die Relation $<$ linear geordnete Menge, und sei r eine natürliche Zahl. Die *lexikographische Ordnung* $<_{lex}$ auf der Menge A^r aller r-Tupel über A ist wie folgt definiert:
Für $\vec{a} = (a_1, .., a_r) \in A^r$ und $\vec{b} = (b_1, .., b_r) \in A^r$ gilt

$$\vec{a} <_{lex} \vec{b} \quad :\Longleftrightarrow \quad \text{es gibt ein } i \in \{1, .., r\}, \text{ so dass } a_i < b_i, \text{ und für alle } j < i \text{ gilt } a_j = b_j.$$

Wie man leicht sieht, ist $<_{lex}$ eine *lineare Ordnung* auf A^r.

1.49 Beispiel. Sei $A = \{0, 1, 2\}$ und sei $<$ die natürliche lineare Ordnung auf A. Für die daraus resultierende lexikographische Ordnung $<_{lex}$ auf A^2 gilt:

$\vec{a} \in A^2$	$(0,0)$	$(0,1)$	$(0,2)$	$(1,0)$	$(1,1)$	$(1,2)$	$(2,0)$	$(2,1)$	$(2,2)$
$rg_{<_{lex}}(\vec{a})$	0	1	2	3	4	5	6	7	8

1.50 Definition (Die Standardkodierung *enc*$(\mathfrak{A})$ einer Struktur $\mathfrak{A}$).
Sei $\sigma = \{R_1, .., R_k, c_1, .., c_\ell\}$ eine Signatur, wobei R_i ein Relationssymbol der Stelligkeit $r_i := ar(R_i)$ und c_i ein Konstantensymbol ist (für alle $i \leqslant k$ bzw. $i \leqslant \ell$).
Sei $r := \max\{r_1, .., r_k\}$ die maximale Stelligkeit eines Relationssymbols in σ.
Sei $\mathfrak{A}$ eine σ-Struktur mit einem Universum A der Mächtigkeit $n := |A|$.

Wir wählen zunächst eine beliebige lineare Ordnung $<$ auf der Menge A, es sei also $A = \{a_0, .., a_{n-1}\}$ mit $a_0 < a_1 < \cdots < a_{n-1}$. Im Folgenden werden wir jedes Element a_i mit seinem Rang $rg_<(a_i)$, also mit der Zahl i identifizieren. Des Weiteren identifizieren wir jedes r_i-Tupel $\vec{a} \in A^{r_i}$ mit seinem Rang $rg_{<_{lex}}(\vec{a}) \in \{0, 1, .., n^{r_i}-1\}$.

Für jedes $i \in \{1, .., k\}$ kodieren wir die Relation $R_i^{\mathfrak{A}}$ durch das Wort

$$enc(R_i^{\mathfrak{A}}) \;:=\; w_0 w_1 \cdots w_{n^r-1} \;\in\; \{0,1\}^{n^r},$$

wobei

- für alle $j \geqslant n^{r_i}$ gilt: $w_j = 0$, und
- für alle $\vec{a} \in A^{r_i}$ gilt: $\vec{a} \in R_i^{\mathfrak{A}} \iff w_{rg_{<_{lex}}(\vec{a})} = 1$.

Konstanten $c_i^{\mathfrak{A}}$ werden analog als Wörter

$$enc(c_i^{\mathfrak{A}}) \;:=\; w_0 w_1 \cdots w_{n^r-1} \;\in\; \{0,1\}^{n^r},$$

kodiert, wobei für alle $j \in \{0, .., n^r-1\}$ gilt:

$$w_j = 1 \iff j = rg_<(c_i^{\mathfrak{A}}).$$

Schließlich kodieren wir die Struktur $\mathfrak{A}$ durch das Wort

$$enc(\mathfrak{A}) \quad := \quad 1^n\, 0^{(n^r-n)}\; enc(R_1^{\mathfrak{A}}) \cdots enc(R_k^{\mathfrak{A}})\; enc(c_1^{\mathfrak{A}}) \cdots enc(c_\ell^{\mathfrak{A}}).$$

1.51 Beispiel. Sei $\sigma = \{R_1, c_1\}$ die Signatur, die aus einem Konstantensymbol c_1 und einem 2-stelligen Relationssymbol R_1 besteht. Wir betrachten die σ-Struktur $\mathfrak{A} = (A, R_1^{\mathfrak{A}}, c_1^{\mathfrak{A}})$ mit $A = \{0, 1, 2\}$, $R_1^{\mathfrak{A}} = \{(0, 1), (1, 1)\}$ und $c_1^{\mathfrak{A}} = 2$. Dann gilt:

$$\mathit{enc}(\mathfrak{A}) \;=\; \underbrace{111000000}_{1^n 0^{n^2-n}}\ \underbrace{010010000}_{\mathit{enc}(R_1^{\mathfrak{A}})}\ \underbrace{001000000}_{\mathit{enc}(c_1^{\mathfrak{A}})}.$$

1.52 Bemerkung. Die Standardkodierung $\mathit{enc}(\mathfrak{A})$ hat folgende Eigenschaften:

(a) Die Kodierung hängt von der gewählten Ordnung des Universums ab.

(b) Wir können schnell auslesen, ob ein Tupel $\vec{a}$ in $R_i^{\mathfrak{A}}$ liegt. Dazu braucht man nur den Schreib-/Lesekopf einer Turing-Maschine in den richtigen Block zu bewegen und dort die $\mathit{rg}_{<_{lex}}(\vec{a})$-te Speicherzelle zu lesen. Insgesamt heißt das:

$$\vec{a} \in R_i^{\mathfrak{A}} \quad \iff \quad \text{auf Bandposition } \ i \cdot n^r + \mathit{rg}_{<_{lex}}(\vec{a}) \ \text{ steht eine 1.}$$

(c) Die Größe der Kodierung, d.h. die Länge des Worts $\mathit{enc}(\mathfrak{A})$, ist polynomiell in der Größe des Universums der Struktur $\mathfrak{A}$ (der Exponent hängt von der Signatur, nicht aber vom Universum A ab). Genauer: $|\mathit{enc}(\mathfrak{A})| = (1 + k + \ell) \cdot |A|^r$.

1.3.2 Datenkomplexität, Programmkomplexität, kombinierte Komplexität

1.53 Definition. Sei $\mathscr{L}$ eine Logik und $\mathscr{K}$ eine Komplexitätsklasse. Wir sagen:

(a) Die *kombinierte Komplexität des Auswertungsproblems für* $\mathscr{L}$ liegt in $\mathscr{K}$, falls gilt:

$$L \;:=\; \{\mathit{enc}(\mathfrak{A})\#\mathit{enc}(\varphi) \mid \mathfrak{A} \in \mathsf{Fin},\ \varphi \in \mathscr{L} \text{ und } \mathfrak{A} \models \varphi\} \;\in\; \mathscr{K}.$$

(Man beachte, dass L eine Menge von Worten über einem endlichen Alphabet ist.)

(b) Die *Datenkomplexität des Auswertungsproblems für* $\mathscr{L}$ liegt in $\mathscr{K}$, falls für alle Sätze $\varphi \in \mathscr{L}$ gilt:

$$L_\varphi \;:=\; \{\mathit{enc}(\mathfrak{A}) \mid \mathfrak{A} \in \mathsf{Fin} \text{ und } \mathfrak{A} \models \varphi\} \;\in\; \mathscr{K}.$$

(c) Die *Programmkomplexität des Auswertungsproblems für* $\mathscr{L}$ liegt in $\mathscr{K}$, falls für alle endlichen Strukturen $\mathfrak{A}$ gilt:

$$L_{\mathfrak{A}} \;:=\; \{\mathit{enc}(\varphi) \mid \varphi \in \mathscr{L} \text{ und } \mathfrak{A} \models \varphi\} \;\in\; \mathscr{K}.$$

1.54 Bemerkung. Bei der *Datenkomplexität* besteht die Eingabe nur aus (der Standardkodierung) einer Struktur $\mathfrak{A}$, während die betrachtete Formel *fest* ist. Die "Komplexität" wird also in der Größe der Struktur gemessen.
Der Name "Datenkomplexität" kommt aus dem Bereich der Datenbanken, in dem die Anfragen (also Formeln) typischerweise klein, die Datenbanken (also Strukturen) hingegen

sehr groß sind. Daher interessiert man sich in diesem Kontext besonders dafür, wie der Ressourcenverbrauch mit der Größe der Datenbank wächst.

Oft ist die *Datenkomplexität* einer Logik viel kleiner als die kombinierte Komplexität, wohingegen kombinierte Komplexität und Programmkomplexität oft zusammenfallen. In dieser Vorlesung werden wir in erster Linie die Datenkomplexität und die kombinierte Komplexität betrachten.

1.55 Definition. Sei $\mathscr{L}$ eine Logik und $\mathscr{K}$ eine Komplexitätsklasse.

(a) Die *kombinierte Komplexität* vom Auswertungsproblem für $\mathscr{L}$ heißt *$\mathscr{K}$-vollständig*, falls die Sprache

$$L := \{enc(\mathfrak{A})\#enc(\varphi) \mid \mathfrak{A} \in \mathsf{Fin},\ \varphi \in \mathscr{L} \text{ und } \mathfrak{A} \models \varphi\}$$

$\mathscr{K}$-vollständig ist.

(b) Die *Datenkomplexität* vom Auswertungsproblem für $\mathscr{L}$ heißt *$\mathscr{K}$-vollständig*, falls sie in $\mathscr{K}$ liegt und es (mindestens) einen Satz $\varphi \in \mathscr{L}$ gibt, so dass die Sprache

$$L_\varphi := \{enc(\mathfrak{A}) \mid \mathfrak{A} \in \mathsf{Fin} \text{ und } \mathfrak{A} \models \varphi\}$$

$\mathscr{K}$-vollständig ist.

(c) Die *Programmkomplexität* vom Auswertungsproblem für $\mathscr{L}$ heißt *$\mathscr{K}$-vollständig*, falls sie in $\mathscr{K}$ liegt und es (mindestens) eine endliche Struktur $\mathfrak{A}$ gibt, so dass die Sprache

$$L_{\mathfrak{A}} := \{enc(\varphi) \mid \varphi \in \mathscr{L} \text{ und } \mathfrak{A} \models \varphi\}$$

$\mathscr{K}$-vollständig ist.

1.3.3 Die Komplexität des Auswertungsproblems für FO

1.56 Lemma. *Es gibt eine deterministische Turing-Maschine, die das Problem*

$$L := \{enc(\mathfrak{A})\#enc(\varphi) \mid \mathfrak{A} \in \mathsf{Fin},\ \varphi \in \mathrm{FO} \textit{ und } \mathfrak{A} \models \varphi\}$$

löst und bei Eingabe eines Worts $enc(\mathfrak{A})\#enc(\varphi)$ *mit* $n := |enc(\mathfrak{A})|$ *und* $m := |enc(\varphi)|$ *nur Platz* $\mathcal{O}\big(m \cdot (\log m + m \cdot \log n)\big)$ *benutzt.*

Beweisidee: Man kann leicht einen rekursiven Algorithmus

$$\text{EVAL}(\mathfrak{A}, \varphi(x_1, \ldots, x_s), a_1, \ldots, a_s)$$

angeben, der bei Eingabe einer endlichen Struktur $\mathfrak{A}$, einer FO-Formel φ und Belegungen $a_1, \ldots, a_s \in A$ für die freien Variablen von φ entscheidet, ob $\mathfrak{A} \models \varphi[a_1, \ldots, a_s]$; siehe z.B.

§ *6.5 Auswertung von Formeln in endlichen Strukturen* in Prof. Grohes Skript zur Vorlesung *Theoretische Informatik I* vom Wintersemester 2004/05, erhältlich unter `http://www.informatik.hu-berlin.de/logik/lehre/WS04-05/ThI1/folien.html`

In jedem rekursiven Aufruf muss dieser Algorithmus sich höchstens folgendes merken:

1. Welche Teilformel ψ von φ wird gerade bearbeitet?

   ... Dazu muss eine DTM sich nur merken, an der wievielten Position von $enc(\varphi)$ diese Teilformel beginnt; es reicht also ein Bitstring der Länge $\log m$.

2. Mit welchen Werten aus dem Universum von $\mathfrak{A}$ werden die freien Variablen von ψ gerade belegt?

   ... Dazu muss eine DTM sich für jede der höchstens m freien Variablen einen Bitstring der Länge $\log n$ merken, der das jeweilige Element im Universum von $\mathfrak{A}$ repräsentiert.

Für jeden Rekursionsschritt wird also höchstens Platz $\mathcal{O}(\log m + m \cdot \log n)$ benötigt.
Da die Rekursionstiefe $\leqslant m$ ist, reicht insgesamt also Platz $\mathcal{O}\big(m \cdot (\log m + m \cdot \log n)\big)$ aus. □

Aus Lemma 1.56 ergibt sich direkt:

1.57 Satz (Die Komplexität des Auswertungsproblems für FO).

(a) Die kombinierte Komplexität des Auswertungsproblems für FO *liegt in* PSPACE.

(b) Die Datenkomplexität des Auswertungsproblems für FO *liegt in* LOGSPACE.

Beweis: Sei M die DTM aus Lemma 1.56, die das Problem

$$L \ := \ \{enc(\mathfrak{A})\#enc(\varphi) \ \mid \ \mathfrak{A} \in \mathsf{Fin}, \ \varphi \in \text{FO und } \mathfrak{A} \models \varphi\}$$

löst und bei Eingaben $enc(\mathfrak{A})\#enc(\varphi)$ mit $n := |enc(\mathfrak{A})|$ und $m := |enc(\varphi)|$ nur Platz $\mathcal{O}\big(m \cdot (\log m + m \cdot \log n)\big)$ benötigt. Wenn N die Länge der gesamten Eingabe bezeichnet, so ist $N = n + m + 1$ und die DTM M ist daher S-platzbeschränkt für eine Funktion $S : \mathbb{N} \to \mathbb{R}_{\geqslant 0}$ mit $S(N) \in$

$$\mathcal{O}\big(N \cdot (\log N + N \cdot \log N)\big) \ = \ \mathcal{O}\big(N \cdot \log N \ + \ N^2 \cdot \log N\big) \ = \ \mathcal{O}\big(N^2 \cdot \log N\big).$$

Somit ist M eine polynomiell platzbeschränkte Turing-Maschine, die das Problem L löst. Also: $L \in$ PSPACE, d.h. die kombinierte Komplexität des Auswertungsproblems für FO liegt in PSPACE. Somit ist *(a)* bewiesen.

Zum Beweis von *(b)* sei φ ein FO-Satz und

$$L_\varphi \ := \ \{enc(\mathfrak{A}) \ \mid \ \mathfrak{A} \in \mathsf{Fin} \text{ und } \mathfrak{A} \models \varphi\}.$$

Man kann leicht eine DTM M_φ bauen, die bei Eingabe *enc*$(\mathfrak{A})$ genau das tut, was M bei Eingabe *enc*$(\mathfrak{A})\#$*enc*(φ) tun würde. M_φ löst dann das Problem L_φ, und da die Formel φ fest ist, ist $m = |enc(\varphi)|$ eine Konstante, d.h., M_φ hat Platzschranke $\mathcal{O}(\log n)$. Für jedes feste $\varphi \in$ FO ist also $L_\varphi \in$ LOGSPACE. Somit liegt die Datenkomplexität des Auswertungsproblems für FO in LOGSPACE. □

Die *Datenkomplexität* des Auswertungsproblems für FO ist tatsächlich noch geringer. Man kann nämlich zeigen, dass sie in einer Schaltkreiskomplexitätsklasse namens AC^0 liegt, von der man weiß, dass $\text{AC}^0 \subsetneq$ LOGSPACE. (Wir werden das in dieser Vorlesung allerdings nicht beweisen.)
Hinsichtlich der *kombinierten Komplexität* ist ein PSPACE-Algorithmus aber das bestmögliche:

1.58 Satz.
Die kombinierte Komplexität des Auswertungsproblems für FO *ist* PSPACE*-vollständig.*

Beweis: Diesen Satz werden wir in Kapitel 4 (Fixpunktlogiken) beweisen. □

1.3.4 Der Satz von Trachtenbrot

Im vorherigen Abschnitt haben wir gesehen, dass das Auswertungsproblem für FO in PSPACE (kombinierte Komplexität) bzw. LOGSPACE (Datenkomplexität) lösbar ist. Nun werden wir uns dem *endlichen Erfüllbarkeitsproblem für* FO zuwenden, also dem Problem

> ERFÜLLBARKEITSPROBLEM FÜR FO$[\sigma]$ IN Fin:
> *Eingabe:* Ein FO$[\sigma]$-Satz φ.
> *Frage:* Gibt es eine σ-Struktur $\mathfrak{A} \in \mathsf{Fin}$ mit $\mathfrak{A} \models \varphi$?

Formal wird dieses Problem durch die folgende Wortsprache kodiert:

$$L_{\text{ERF-FO}[\sigma]\text{-}\mathsf{Fin}} := \left\{ enc(\varphi) \;\middle|\; \begin{array}{l} \varphi \text{ ist ein FO}[\sigma]\text{-Satz, für den es eine } \sigma\text{-Struktur} \\ \mathfrak{A} \in \mathsf{Fin} \text{ mit } \mathfrak{A} \models \varphi \text{ gibt} \end{array} \right\}.$$

1.59 Theorem (Satz von Trachtenbrot, 1950).
Es gibt eine endliche Signatur σ, so dass das Erfüllbarkeitsproblem für FO$[\sigma]$ *in* Fin *unentscheidbar ist.*

Beweis: Wir müssen zeigen, dass es keine Turing-Maschine geben kann, die das Problem $L_{\text{ERF-FO}[\sigma]\text{-}\mathsf{Fin}}$ entscheidet, d.h., die bei Eingabe der Kodierung eines FO$[\sigma]$-Satzes entscheidet, ob dieser ein endliches Modell hat.

Idee: Wir wissen, dass das Halteproblem auf leerem Eingabewort, H_ε, unentscheidbar ist (siehe Satz 1.19). Wir kodieren nun die Berechnung einer beliebigen Turing-Maschine M (von der wir wissen wollen, ob sie bei leerer Eingabe anhält) durch einen FO-Satz φ_M, der genau dann ein *endliches* Modell hat, wenn die Berechnung von M bei leerem Eingabewort