How to Measure Query Answering Complexity

Query answering as decision problem
\( \sim \) consider Boolean queries

Various notions of complexity:
- Combined complexity (complexity w.r.t. size of query and database instance)
- Data complexity (worst case complexity for any fixed query)
- Query complexity (worst case complexity for any fixed database instance)

Various common complexity classes:
- \( L \subseteq NL \subseteq P \subseteq NP \subseteq \text{PSpace} \subseteq \text{ExpTime} \)

An Algorithm for Evaluating FO Queries

\[
\text{function Eval}(\psi, I) \\
\begin{cases}
01 & \text{switch } (\psi) \{ \\
02 & \quad \text{case } p(c_1, \ldots, c_n) : \text{return } (c_1, \ldots, c_n) \in p^I \\
03 & \quad \text{case } \neg \phi : \text{return } \neg \text{Eval}(\phi, I) \\
04 & \quad \text{case } \phi_1 \land \phi_2 : \text{return } \text{Eval}(\phi_1, I) \land \text{Eval}(\phi_2, I) \\
05 & \quad \text{case } \exists x. \phi : \\
06 & \quad \quad \text{for } c \in \Delta^I \{ \\
07 & \quad \quad \quad \text{if } \text{Eval}(\phi[x \mapsto c], I) \text{ then return true} \\
08 & \quad \quad \} \\
09 & \quad \text{return false} \}
\end{cases}
\]

FO Algorithm Worst-Case Runtime

Let \( m \) be the size of \( \psi \), and let \( n = |I| \) (total table sizes)
**FO Algorithm Worst-Case Runtime**

Let $m$ be the size of $\varphi$, and let $n = |I|$ (total table sizes)

- How many recursive calls of Eval are there? 
  $\sim$ one per subexpression: at most $m$
- Maximum depth of recursion? 
  $\sim$ bounded by total number of calls: at most $m$
- Maximum number of iterations of for loop? 
  $\sim |\Delta^i| \leq n$ per recursion level 
  $\sim$ at most $n^m$ iterations
- Checking $\langle c_1, \ldots, c_n \rangle \in p^I$ can be done in linear time w.r.t. $n$

**Time Complexity of FO Algorithm**

Let $m$ be the size of $\varphi$, and let $n = |I|$ (total table sizes)

Runtime in $m \cdot n^m \cdot n = m \cdot n^{m+1}$

**Space Complexity of FO Algorithm**

Let $m$ be the size of $\varphi$, and let $n = |I|$ (total table sizes)

Memory in $m \log m + (m + 1) \log n$

**FO Algorithm Worst-Case Memory Usage**

We can get better complexity bounds by looking at memory

Let $m$ be the size of $\varphi$, and let $n = |I|$ (total table sizes)

- For each (recursive) call, store pointer to current subexpression of $\varphi$: $\log m$
- For each variable in $\varphi$ (at most $m$), store current constant assignment (as a pointer): $m \cdot \log n$
- Checking $\langle c_1, \ldots, c_n \rangle \in p^I$ can be done in logarithmic space w.r.t. $n$

Memory in $m \log m + m \log n + \log n = m \log m + (m + 1) \log n$
The algorithm shows that FO query evaluation is in PSpace.

**Is this the best we can get?**

**Hardness proof:** reduce a known PSpace-hard problem to FO query evaluation

---

**PSpace-hardness for DI Queries**

The previous reduction from QBF may lead to a query that is not domain independent.

**Example:** QBF $3p, \neg p$ leads to FO query $\exists x, \neg \text{true}(x)$

---

**Better approach:**
- Consider QBF $Q_1 x_1, Q_2 x_2, \ldots, Q_n x_n \varphi(x_1, \ldots, x_n)$ with $\varphi$ in negation normal form: negations only occur directly before variables $X_i$ (still PSpace-complete: exercise)
- Database instance $I$ with $\Delta^2 = \{0, 1\}$
- Two tables with one row each: $\text{true}(1)$ and $\text{false}(0)$
- Transform input QBF into Boolean FO query

$$Q_1 x_1, Q_2 x_2, \ldots, Q_n x_n \varphi'$$

where $\varphi'$ is obtained by replacing each negated variable $\neg X_i$ with $\text{false}(x_i)$ and each non-negated variable $X_i$ with $\text{true}(x_i)$. 

---

**PSpace-hardness for DI Queries**

The previous reduction from QBF may lead to a query that is not domain independent.

**Example:** QBF $3p, \neg p$ leads to FO query $\exists x, \neg \text{true}(x)$

---
Summary and Outlook

The evaluation of FO queries is
- PSpace-complete for combined complexity
- PSpace-complete for query complexity

Open questions:
- What is the data complexity of FO queries?
- Are there query languages with lower complexities? (next lecture)
- Which other computing problems are interesting?