How to Measure Query Answering Complexity

Query answering as decision problem
\rightarrow 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:

\[ L \subseteq NL \subseteq P \subseteq NP \subseteq PSpace \subseteq ExpTime \]

An Algorithm for Evaluating FO Queries

```plaintext
function Eval(ϕ, I)
01 switch (ϕ) {
02 case p(c₁, ..., cₙ) : return ⟨c₁, ..., cₙ⟩ ∈ p(^I)
03 case ¬ψ : return ¬Eval(ψ, I)
04 case ψ₁ ∧ ψ₂ : return Eval(ψ₁, I) ∧ Eval(ψ₂, I)
05 case ∃x.ψ :
06 for c ∈ Δ^I {
07 if Eval(ψ[x → c], I) then return true
08 }
09 return false
10 }
```

FO Algorithm Worst-Case Runtime

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

- How many recursive calls of Eval are there?
  \rightarrow one per subexpression: at most \( m \)
- Maximum depth of recursion?
  \rightarrow bounded by total number of calls: at most \( m \)
- Maximum number of iterations of for loop?
  \rightarrow |Δ^I| ≤ n per recursion level
  \rightarrow at most \( n^m \) iterations
- Checking \( ⟨c₁, ..., cₙ⟩ ∈ p^I \) can be done in linear time w.r.t. \( n \)

Runtime in \( m \cdot n^m \cdot n = m \cdot n^{m+1} \)
Time Complexity of FO Algorithm

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

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

- Time complexity of FO query evaluation
  - Combined complexity: in ExpTime
  - Data complexity ($m$ is constant): in P
  - Query complexity ($n$ is constant): in ExpTime

Space Complexity of FO Algorithm

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

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

- Space complexity of FO query evaluation
  - Combined complexity: in PSpace
  - Data complexity ($m$ is constant): in L
  - Query complexity ($n$ is constant): in PSpace

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 $m \log m + m \log n + \log n = m \log m + (m + 1) \log n$

FO Combined Complexity

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 $\leadsto$ QBF satisfiability

Let $O_1 X_1, O_2 X_2, \ldots, O_n X_n, \varphi[X_1, \ldots, X_n]$ be a QBF (with $O_i \in \{Y, \exists\}$)

- Database instance $I$ with $\Delta^I = \{0, 1\}$
- One table with one row: $true(1)$
- Transform input QBF into Boolean FO query

\[
O_1 x_1, O_2 x_2, \ldots, O_n x_n, \varphi[X_1 \mapsto true(x_1), \ldots, X_n \mapsto true(x_n)]
\]

It is easy to check that this yields the required reduction. \(\square\)
PSpace-hardness for DI Queries

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

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

**Better approach:**
- Consider QBF $Q_1 X_1. Q_2 X_2. \cdots Q_n X_n. \varphi[X_1, \ldots, X_n]$ with $\varphi$ in negation normal form: negations only occur directly before variables $X_i$ (still PSpace-complete: exercise)
- Database instance $I$ with $\Delta_I^f = \{0, 1\}$
- Two tables with one row each: true(1) and false(0)
- Transform input QBF into Boolean FO query $Q_1 x_1. Q_2 x_2. \cdots Q_n x_n. \varphi'$

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

Combined Complexity of FO Query Answering

Summing up, we obtain:

**Theorem 4.1:** The evaluation of FO queries is PSpace-complete with respect to combined complexity.

We have actually shown something stronger:

**Theorem 4.2:** The evaluation of FO queries is PSpace-complete with respect to query complexity.

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?