TECHNISCHE UNIVERSITÄT DRESDEN

DATABASE THEORY

Lecture 4: Complexity of FO Query Answering

Markus Krötzsch

Knowledge-Based Systems

TU Dresden, 23rd Apr 2019

How to Measure Query Answering Complexity

Query answering as decision problem
$\leadsto$ 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:

$$\text{L} \subseteq \text{NL} \subseteq \text{P} \subseteq \text{NP} \subseteq \text{PSpace} \subseteq \text{ExpTime}$$

An Algorithm for Evaluating FO Queries

function Eval($\varphi, \mathcal{I}$)

```
switch (φ) {
    case p(c_1, ..., c_n) : return ⟨c_1, ..., c_n⟩ ∈ p^I
    case ¬ψ : return ¬Eval(ψ, I)
    case ψ_1 ∧ ψ_2 : return Eval(ψ_1, I) ∧ Eval(ψ_2, I)
    case ∃x.ψ :
        for c ∈ Δ^I {
            if Eval(ψ[x ↦ c], I) then return true
        }
        return false
}
```

FO Algorithm Worst-Case Runtime

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

FO Algorithm Worst-Case Runtime

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

- How many recursive calls of Eval are there?
  $\leadsto$ one per subexpression: at most m
- Maximum depth of recursion?
  $\leadsto$ bounded by total number of calls: at most m
- Maximum number of iterations of **`for`** loop?
  $\leadsto$ $|\Delta^{\mathcal{I}}| \leq n$ per recursion level
  $\leadsto$ at most n^m iterations
- Checking $\langle c_1, \ldots, c_n \rangle \in p^{\mathcal{I}}$ 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 φ, and let $n = |\mathcal{I}|$ (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

FO Algorithm Worst-Case Memory Usage

We can get better complexity bounds by looking at memory

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

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

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

Space Complexity of FO Algorithm

Let m be the size of φ, and let $n = |\mathcal{I}|$ (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 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

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
$\rightsquigarrow$ QBF satisfiability

Let $Q_1X_1.Q_2X_2.\cdots Q_nX_n.\varphi[X_1,\ldots,X_n]$ be a QBF (with $Q_i \in \{\forall,\exists\}$)

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

$$Q_1x_1.Q_2x_2.\cdots Q_nx_n.\varphi[X_1 \mapsto \mathsf{true}(x_1),\ldots,X_n \mapsto \mathsf{true}(x_n)]$$

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 $\exists x.\neg\text{true}(x)$

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 $\exists x.\neg\mathsf{true}(x)$

Better approach:

- Consider QBF $\mathsf{Q}_1 X_1.\mathsf{Q}_2 X_2.\cdots\mathsf{Q}_n X_n.\varphi[X_1,\ldots,X_n]$ with φ in negation normal form: negations only occur directly before variables X_i (still PSpace-complete: exercise)
- Database instance $\mathcal{I}$ with $\Delta^{\mathcal{I}} = \{0,1\}$
- Two tables with one row each: true(1) and false(0)
- Transform input QBF into Boolean FO query

$$\mathsf{Q}_1 x_1.\mathsf{Q}_2 x_2.\cdots\mathsf{Q}_n x_n.\varphi'$$

where φ' 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.

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?