Overview

1. Introduction | Relational data model
2. First-order queries
3. Complexity of query answering
4. Complexity of FO query answering
5. Query optimization
6. Conjunctive queries
7. Limits of first-order query expressiveness
8. Introduction to Datalog
9. Implementation techniques for Datalog
10. Path queries
11. Constraints (1)
12. Constraints (2)
13. “Buffer time”
14. Outlook: database theory in practice

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

\[
\text{function } \text{Eval}(\varphi, I) \\
01 \quad \text{switch } (\varphi) \{ \\
02 \quad \text{case } p(c_1, \ldots, c_n) : \text{return } \langle c_1, \ldots, c_n \rangle \in p^I \\
03 \quad \text{case } \neg \psi : \text{return } \neg \text{Eval}(\psi, I) \\
04 \quad \text{case } \psi_1 \land \psi_2 : \text{return } \text{Eval}(\psi_1, I) \land \text{Eval}(\psi_2, I) \\
05 \quad \text{case } \exists x. \psi : \\
06 \quad \quad \text{for } c \in \Delta^I \{ \\
07 \quad \quad \quad \text{if } \text{Eval}(\psi[x \mapsto c], I) \text{ then return true } \\
08 \quad \quad \} \\
09 \quad \text{return } \text{false} \\
10 \} 
\]
**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?
  $\leadsto$ 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?
  $\leadsto |\Delta^I| \leq n$ per recursion level
  $\leadsto$ 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. $n$

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 $m \cdot n^m+1$

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

**Space Complexity of FO Algorithm**

Space complexity of FO query evaluation

- Combined complexity: in PSPACE
- Data complexity ($m$ is constant): in L
- Query complexity ($n$ is constant): in PSPACE
The algorithm shows that FO query evaluation is in \textit{PSpace}. Is this the best we can get?

\textbf{Hardness proof:} reduce a known \textit{PSpace}-hard problem to FO query evaluation

\[ \sim \text{QBF satisfiability} \]

Let \( \forall_1 x_1. \forall_2 x_2. \cdots \forall_n x_n. \varphi(x_1, \ldots, x_n) \) be a QBF (with \( \forall_i \in \{ \forall, \exists \} \))

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

\[ \forall_1 x_1. \forall_2 x_2. \cdots \forall_n x_n. \varphi \leftrightarrow \text{true}(x_1), \ldots, \text{true}(x_n) \]

\textbf{Combined Complexity of FO Query Answering}

\textbf{Theorem}

The evaluation of FO queries is \textit{PSpace}-complete with respect to combined complexity.

We have actually shown something stronger:

\textbf{Theorem}

The evaluation of FO queries is \textit{PSpace}-complete with respect to query complexity.

\textbf{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 \( \forall_1 X_1. \forall_2 X_2. \cdots \forall_n X_n. \varphi(X_1, \ldots, X_n) \) with \( \varphi \) in negation normal form: negations only occur directly before variables \( X_i \) (still \textit{PSpace}-complete: exercise)
- Database instance \( I \) with \( \Delta^I = \{ 0, 1 \} \)
- Two tables with one row each: \text{true}(1) and \text{false}(0)
- Transform input QBF into Boolean FO query

\[ \forall_1 X_1. \forall_2 X_2. \cdots \forall_n X_n. \varphi' \]

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).
**Boolean Circuits**

**Definition**

A **Boolean circuit** is a finite, directed, acyclic graph where:

- each node that has no predecessors is an input node
- each node that is not an input node is one of the following types of logical gate: AND, OR, NOT
- one or more nodes are designated output nodes

We will only consider Boolean circuits with exactly one output.

Propositional logic formulae are Boolean circuits with one output and gates of fanout $\leq 1$.

**Example**

A Boolean circuit over an input string $x_1x_2 \ldots x_n$ of length $n$:

- $n^2$ processors working in parallel
- computation finishes in 2 steps

**Circuits as a Model for Parallel Computation**

Previous example:

- $n^2$ processors working in parallel
- computation finishes in 2 steps

- size: number of gates = total number of computing steps
- depth: longest path of gates = time for parallel computation

- refinement of polynomial time taking parallelizability into account

**Solving Problems With Circuits**

Observation: the input size is “hard-wired” in circuits
- each circuit only has a finite number of different inputs
- not a computationally interesting problem

How can we solve interesting problems with Boolean circuits?

**Definition**

A **uniform family** of Boolean circuits is a set of circuits $C_n$ ($n \geq 0$) that can be computed from $n$ (usually in logarithmic space or time; we don’t discuss the details here).

A language $L \subseteq \{0, 1\}^*$ is decided by a uniform family $(C_n)_{n \geq 0}$ of Boolean circuits if for each word $w$ of length $|w|$: $w \in L$ if and only if $C_{|w|}(w) = 1$.
Measuring Complexity with Boolean Circuits

How to measure the computing power of Boolean circuits?

Relevant metrics:
- size of the circuit: overall number of gates (as function of input size)
- depth of the circuit: longest path of gates (as function of input size)
- fan in: two inputs per gate or any number of inputs per gate?

Important classes of circuits: small-depth circuits

Definition
$(C_n)_{n \geq 0}$ is a family of small-depth circuits if
- the size of $C_n$ is polynomial in $n$,
- the depth of $C_n$ is poly-logarithmic in $n$, that is, $O(\log^k n)$.

The Complexity Classes $NC$ and $AC$

Two important types of small-depth circuits

Definition
$NC^k$ is the class of problems that can be solved by uniform families of circuits $(C_n)_{n \geq 0}$ of fan-in $\leq 2$, size polynomial in $n$, and depth in $O(\log^k n)$.

The class $NC$ is defined as $NC = \bigcup_{k \geq 0} NC^k$.

(*Nick’s Class* named after Nicholas Pippenger by Stephen Cook)

Definition
$AC^k$ and $AC$ are defined like $NC^k$ and $NC$, respectively, but for circuits with arbitrary fan-in.

(A is for *Alternating*: AND-OR gates alternate in such circuits)

Example

family of polynomial size, constant depth, arbitrary fan-in circuits
$\leadsto$ in $AC^0$

We can eliminate arbitrary fan-ins by using more layers of gates:

family of polynomial size, logarithmic depth, bounded fan-in circuits
$\leadsto$ in $NC^1$

Relationships of Circuit Complexity Classes

The previous sketch can be generalised:

$NC^0 \subseteq AC^0 \subseteq NC^1 \subseteq AC^1 \subseteq \ldots \subseteq AC^k \subseteq NC^{k+1} \subseteq \ldots$

Only few inclusions are known to be proper: $NC^0 \subset AC^0 \subset NC^1$

Direct consequence of above hierarchy: $NC = AC$

Interesting relations to other classes:

$NC^0 \subset AC^0 \subset NC^1 \subset L \subset NL \subset AC^1 \subset \ldots \subset NC \subset P$

Intuition:
- Problems in $NC$ are parallelisable
- Problems in $P \setminus NC$ are inherently sequential

However: it is not known if $NC \neq P$
The evaluation of FO queries is complete for (logtime uniform) AC0 with respect to data complexity.

Proof:
- **Membership:** For a fixed Boolean FO query, provide a uniform construction for a small-depth circuit based on the size of a database.
- **Hardness:** Show that circuits can be transformed into Boolean FO queries in logarithmic time (not on a standard TM . . . not in this lecture).

Assumption:
- query and database schema is fixed
- database instance (and thus active domain) are variable

Construct circuit uniformly based on size of active domain.

Sketch of construction:
- one input node for each possible database tuple (over given schema and active domain)
- Recursively, for each subformula, introduce a gate for each possible tuple (instantiation) of this formula
- Logical operators correspond to gate types: basic operators obvious, ∀ as generalised conjunction, ∃ as generalised disjunction
- subformula with \(n\) free variables \(\leadsto\) \(|\text{adom}|^{n}\) gates
- especially: \(|\text{adom}|^{0} = 1\) output gate for Boolean query

Example:

We consider the formula

\[ \exists z. (\exists x. \exists y. R(x, y) \land S(y, z)) \land \neg R(a, z) \]

Over the database instance:

<table>
<thead>
<tr>
<th>R</th>
<th>S</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>b</td>
</tr>
<tr>
<td>a</td>
<td>b</td>
</tr>
</tbody>
</table>

Active domain: \(\{a, b, c\}\)
Example: $\exists z. (\exists x. \exists y. R(x, y) \land S(y, z)) \land \neg R(a, z)$

Example: $\exists x. (\exists y. R(x, y) \land S(y, z)) \land \neg R(a, z)$

Summary and Outlook

The evaluation of FO queries is
- **PSpace-complete** for combined complexity
- **PSpace-complete** for query complexity
- **AC^0**-complete for data complexity

Circuit complexities help to identify highly parallelisable problems in P

Open questions:
- Which other computing problems are interesting? (next lecture)
- Are there query languages with lower complexities?
- How can we study the expressiveness of query languages?