How to Measure Query Answering Complexity

Query answering as decision problem
\(~\) 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:
\[ L \subseteq NL \subseteq P \subseteq NP \subseteq PSpace \subseteq ExpTime \]
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 $m \cdot n^m \cdot n = m \cdot n^{m+1}$

Markus Krötzsch, 21 April 2016

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

Markus Krötzsch, 21 April 2016

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

Markus Krötzsch, 21 April 2016

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

Markus Krötzsch, 21 April 2016
The algorithm shows that FO query evaluation is in \( \text{PSpace} \).
Is this the best we can get?

Hardness proof: reduce a known \( \text{PSpace} \)-hard problem to FO query evaluation
\( \leadsto \) QBF satisfiability

Let \( Q_1 X_1. Q_2 X_2. \cdots Q_n X_n. \varphi[X_1, \ldots, X_n] \) be a QBF (with \( Q_i \in \{\forall, \exists\} \))

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

\[
Q_1 x_1. Q_2 x_2. \cdots Q_n x_n. \varphi'[x_1 \mapsto \text{true}(x_1), \ldots, x_n \mapsto \text{true}(x_n)]
\]

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 \( \text{PSpace} \)-complete: exercise)
- Database instance \( I \) with \( \Delta^I = \{0, 1\} \)
- 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(\( x_i \)).
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 ≤ 1

---

Circuits as a Model for Parallel Computation

**Previous example:**

→ \( n^2 \) processors working in parallel
→ computation finishes in 2 steps

→ 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 \( \mathcal{L} \subseteq \{0, 1\}^* \) is decided by a uniform family \( (C_n)_{n \geq 0} \) of Boolean circuits if for every word \( w \) of length \(|w|\):

\[ w \in \mathcal{L} \text{ if and only if } C_{|w|}(w) = 1 \]
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

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:

\(NC^0 \subset AC^0 \subset NC^1 \subseteq L \subseteq NL \subseteq AC^1 \subseteq \ldots \subseteq NC \subseteq P\)

Intuition:

- Problems in \(NC\) are parallelisable
- Problems in \(P \setminus NC\) are inherently sequential

However: it is not known if \(NC \neq P\)
Theorem

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).

**From Query to Circuit**

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)
  \( \leadsto \) true or false depending on whether tuple is present or not

• Recursively, for each subformula, introduce a gate for each possible tuple (instantiation) of this formula
  \( \leadsto \) true or false depending on whether the subformula holds for this tuple or not

• Logical operators correspond to gate types: basic operators obvious, \( \forall \) as generalised conjunction, \( \exists \) as generalised disjunction

• subformula with \( n \) free variables \( \leadsto \) \( |\text{adom}| \cdot n \) gates
  \( \leadsto \) especially: \( |\text{adom}| = 1 \) 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**: \( \exists z. (\exists x. \exists y. R(x, y) \land S(y, z)) \land \neg R(a, z) \)
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?