Review: Query 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:
\[ L \subseteq NL \subseteq P \subseteq NP \subseteq PSpace \subseteq ExpTime \]

Review: FO Combined Complexity

Theorem 4.1 The evaluation of FO queries is \( \text{PSpace} \)-complete with respect to combined complexity.

We have actually shown something stronger:

Theorem 4.2 The evaluation of FO queries is \( \text{PSpace} \)-complete with respect to query complexity.

This also holds true when restricting to domain-independent queries.

Data Complexity of FO Query Answering

The algorithm showed that FO query evaluation is in \( L \)
\[ \leadsto \]
- can we do any better?

What could be better than \( L \)?

\[ ? \subseteq L \subseteq NL \subseteq P \subseteq \ldots \]

\[ \leadsto \] we need to define circuit complexities first
Boolean Circuits

Definition 5.1: 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

Example
A Boolean circuit over an input string \( x_1x_2 \ldots x_n \) of length \( n \)
\[ \ldots \text{(} n^2 \text{ gates) \ldots} \]
Corresponds to formula \((x_1 \land x_2) \lor (x_1 \land x_3) \lor \ldots \lor (x_{n-1} \land x_n)\)
→ accepts all strings with at least two 1s

Circuits as a Model for Parallel Computation

Previous example:
\[ \ldots \]
\( \ldots \text{ processors working in parallel} \)
\( \ldots \text{ computation finishes in 2 steps} \)

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

→ circuits as a refinement of polynomial time that takes 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 5.2: A uniform family of Boolean circuits is a set of circuits \( C_n \) (\( n \geq 0 \)) that can easily be computed from \( n \).

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

\( ^a \)We don’t discuss the details here; see course Complexity Theory.
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 5.3: \((C_n)_{n \geq 0}\) 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 5.4: \(NC^k\) 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 5.5: \(AC^k\) 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:

\[ NC^0 \subset AC^0 \subset NC^1 \subseteq \mathcal{L} \subseteq \mathcal{NL} \subseteq AC^1 \subseteq \ldots \subseteq \mathcal{NC} \subseteq \mathcal{P} \]

Intuition:
- Problems in NC are parallelisable (known from definition)
- Problems in \(P \setminus NC\) are inherently sequential (educated guess)

However: It is not known if \(NC \neq P\)
Theorem 5.6: The evaluation of FO queries is complete for (logtime uniform) AC$^0$ 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

**Assumptions:**
- 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, $\forall$ as generalised conjunction, $\exists$ as generalised disjunction
- subformula with $n$ free variables $\leadsto$ $|\text{adom}|^n$ gates
  $\leadsto$ 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>a</td>
</tr>
<tr>
<td>a</td>
<td>b</td>
</tr>
<tr>
<td>a</td>
<td>c</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
- AC⁰-complete for data complexity

Circuit complexities help to identify highly parallelisable problems in P

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