Review

We have studied FO queries and the simpler conjunctive queries. Our focus was on query answering complexity:

<table>
<thead>
<tr>
<th>Type of Query</th>
<th>Combined Complexity</th>
<th>Query Complexity</th>
<th>Data Complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>FO queries</td>
<td>PSpace-comp.</td>
<td>PSpace-comp.</td>
<td>in AC₀</td>
</tr>
<tr>
<td>Conjunctive queries</td>
<td>NP-comp.</td>
<td>NP-comp.</td>
<td>in AC₀</td>
</tr>
<tr>
<td>Tree CQs</td>
<td>in P</td>
<td>in P</td>
<td>in AC₀</td>
</tr>
<tr>
<td>Bounded Treewidth CQs</td>
<td>in P</td>
<td>in P</td>
<td>in AC₀</td>
</tr>
<tr>
<td>Bounded Hypertree width CQs</td>
<td>in P</td>
<td>in P</td>
<td>in AC₀</td>
</tr>
</tbody>
</table>

Static Query Optimisation

Can we optimise query execution without looking at the database?

Queries are logical formulae, so some things might follow . . .

**Query equivalence:**
Will the queries $Q_1$ and $Q_2$ return the same answers over any database?
- In symbols: $Q_1 \equiv Q_2$
- We have seen many examples of equivalent transformations in exercises
- Several uses for optimisation:
  - $\leadsto$ DBMS could run the “nicer” of two equivalent queries
  - $\leadsto$ DBMS could use cached results of one query for the other
  - $\leadsto$ Also applicable to equivalent subqueries

**Static Query Optimisation (2)**

Other things that could be useful:
- **Query emptiness:** Will query $Q$ never have any results?
  - $\leadsto$ Special equivalence with an “empty query”
    - (e.g., $\emptyset \equiv x \neq x \lor R(x) \land \neg R(x)$)
  - $\leadsto$ Empty (sub)queries could be answered immediately
- **Query containment:** Will the query $Q_1$ return a subset of the results of query $Q_2$?
  - (in symbols: $Q_1 \subseteq Q_2$)
  - $\leadsto$ Generalisation of equivalence:
    - $Q_1 \equiv Q_2$ if and only if $Q_1 \subseteq Q_2$ and $Q_2 \subseteq Q_1$
- **Query minimisation:** Given a query $Q$, can we find an equivalent query $Q'$ that is “as simple as possible.”
First-order logic: Decidable or not?

We have seen in recent lectures:
- FO queries can be answered in PSpace (combined complexity) and AC$^0$ (data complexity)
- FO queries correspond to relational algebra, so every relational DBMS answers FO queries in practice

In foundational courses on logic, you should have learned
- Reasoning in first-order logic is undecidable

Indeed, Wikipedia says it too (so it must be true . . .):
- "Unlike propositional logic, first-order logic is undecidable (although semidecidable)" [Wikipedia article First-order logic]

Is the first-order logic we use different from the first-order logic used elsewhere?
Is mathematics inconsistent?

Solving the Mystery

All of the above are true for first-order logic but people are studying different decision problems:

Problem 1: Model Checking
- Given: a logical sentence $\varphi$ and a finite model $I$
- Question: is $I$ a model for $\varphi$, i.e., is $\varphi$ satisfied in $I$?
- Corresponds to Boolean query entailment
- PSpace-complete for first-order sentences

Problem 2: Satisfiability Checking
- Given: a logical sentence $\varphi$
- Question: does $\varphi$ have any model?
- (Turing-)equivalent to many reasoning problems (entailment, tautology, unsatisfiability, etc.)
- Undecidable for first-order sentences

Back to Query Optimisation

What do these results mean for query optimisation?

Two similar questions:
1. Are the Boolean FO queries $\varphi_1$ and $\varphi_2$ equivalent?
2. Are the FO sentences $\varphi_1$ and $\varphi_2$ equivalent?

$\not\sim$ So FO query equivalence is undecidable?

However, (1) is not equivalent to (2) but to the following:
2'. Are the FO sentences $\varphi_1$ and $\varphi_2$ equivalent in all finite interpretations?
$\not\sim$ finite-model reasoning for FO logic

Finite-Model Reasoning

Does it really make a difference?

Yes. Example formula $\varphi$:

\[
\forall x : (\exists y \forall z : R(x, y) \land R(x, y) \land R(x, y)) \land \\
(\forall x, y_1, y_2 : R(x, y_1) \land R(x, y_2) \rightarrow y_1 = y_2) \land \\
(\forall x_1, x_2, y : R(x_1, y) \land R(x_2, y) \rightarrow x_1 = x_2) \land \\
(\exists y : \forall x : \neg R(x, y))
\]

\[
R \text{ is a function . . .} \\
\text{and injective . . .} \\
\text{but not surjective}
\]

Such a function $R$ can only exist over an infinite domain.

$\not\sim$ over finite models, $\varphi$ is unsatisfiable
$\not\sim$ $\varphi$ is finitely equivalent to $\forall x : \exists y : R(x, y) \land \neg R(x, y)$
$\not\sim$ this equivalence does not hold on arbitrary models
Trakhtenbrot's Theorem

Is finite-model reasoning easier than FO reasoning in general?

Unfortunately no:

**Theorem 9.1 (Boris Trakhtenbrot, 1950):** Finite-model reasoning of first-order logic is undecidable.

Interesting observation:
- The set of all true sentences (tautologies) of FO is recursively enumerable ("FO entailment is semi-decidable")
- but the set of all FO tautologies under finite models is not.

$\sim$ finite model reasoning is harder than FO reasoning in this case!

Let's Prove Trakhtenbrot's Theorem

**Proof idea:** reduce the Halting Problem to finite satisfiability

- Input of the reduction: a deterministic Turing Machine (DTM) $M$ and an input string $w$
- Output of the reduction: a first-order formula $\varphi_{M,w}$
- Such that $M$ halts on $w$ if and only if $\varphi_{M,w}$ has a finite model

Ok, this would do, because Halting of DTMs is undecidable, but how should we achieve this?
- Capture the computation of the DTM in a finite model
- The model contains the whole run: the tape and state for every computation step
- A finite part of the tape is enough if the DTM halts

TM Runs as Finite Models

**Recall:** Turing Machine is given as $M = \langle Q, q_{\text{start}}, q_{\text{acc}}, \Sigma, \Lambda \rangle$

(state set $Q$, tape alphabet $\Sigma$ with blank $\$", transitions $\Lambda \subseteq (Q \times \Sigma) \times (Q \times \Sigma \times \{l, r, s\})$)

A configuration is a (finite piece of) tape + a position + a state:

$q \in Q$

$\downarrow$

TAPE CONTENTS ...

Here is how we want part of our model (database) to look:

$S_T \quad S_s \quad S_P \quad S_E \quad H_q \quad S_C$

right right right right right

Encoding TM Runs as Relational Structures

We use several **unary predicate symbols** to mark tape cells:
- $S_T(\cdot)$ for each $\sigma \in \Sigma$: tape cell contains symbol $\sigma$
- $H_q(\cdot)$ for each $q \in Q$: head is at tape cell, and TM is in state $q$

We use two **binary predicate symbols** to connect tape positions:
- $\text{right}(\cdot, \cdot)$: neighbouring tape cells at same step
- $\text{right}^+(\cdot, \cdot)$: transitive super-relation of right
- $\text{future}(\cdot, \cdot)$: tape cells at same position in consecutive steps
The TM can only be in one state:

The TM is never at more than one position:

A cell can only contain one symbol:

We now need to specify formulae to enforce this intended structure (or something that is close enough to it).

Consistent Tape Contents, Head, and State

A cell can only contain one symbol:

The TM is never at more than one position:

The TM can only be in one state:

Defining the Initial Configuration

Require that right* is a transitive super-relation of right:

Define start configuration for an input word w = \sigma_1 \sigma_2 \ldots \sigma_n:

... where there can be any number of cells right of the input, but they must contain ...

Transitions

For every non-moving transition \delta = (q, \sigma, q', \sigma', a) \in \Delta:

For every right-moving transition \delta = (q, \sigma, q', \sigma', r) \in \Delta:

For every left-moving transition \delta = (q, \sigma, q', \sigma', l) \in \Delta:

Summing all up:

\psi_{\Delta} = \bigwedge_{\delta \in \Delta} \psi_{\Delta}
Preserve Tape if not Changed by Transition

Contents of tape cells that are not under the head are kept:

$$\varphi_{\text{mem}} = \forall x, y. \bigwedge_{n \in \mathbb{Z}} S_n(x) \wedge \left( \bigwedge_{\varphi \in \mathcal{Q}} \neg H_{\varphi}(x) \wedge \text{future}(x, y) \rightarrow S_n(y) \right)$$

Finishing the Proof of Trakhtenbrot’s Theorem

We obtain a final FO formula

$$\psi_{M,w} = \psi_{\text{right}} \land \psi_{\text{left}} \land \psi_{\text{left}} \land \psi_{\text{right}} \land \psi_{\text{left}} \land \psi_{\text{mem}} \land \psi_{p0} \land \psi_{p1} \land \psi_{l} \land \psi_{\text{right}} \land \psi_{\text{left}}$$

Then $\psi_{M,w}$ is finitely satisfiable if and only if $M$ halts on $w$:

- If $M$ has a finite run when started on $w$, then $\psi_{M,w}$ has a finite model that encodes this run.
- If $\psi_{M,w}$ has a finite model, then we can extract from this model a finite run of $M$ on $w$.

Note: the proof can be made to work using only one binary relation symbol and no equality (not too hard, but less readable)

Building the Configuration Grid

If one cell has a future ($\to$) or past ($\leftarrow$), respectively, all cells of the tape do:

$$\varphi_{l1} = \exists x_2, y_2. (\exists x_1. \text{right}(x_1, y_1) \land \text{future}(x_1, x_2)) \leftrightarrow (\exists y_2. \text{future}(y_1, y_2) \land \text{right}(x_2, y_2))$$

$$\varphi_{r2} = \exists x_1, y_2. (\exists y_1. \text{right}(x_1, y_1) \land \text{future}(y_1, y_2)) \leftrightarrow (\exists x_2. \text{future}(x_1, x_2) \land \text{right}(y_2, y_2))$$

Left ($l$) and right ($r$) neighbours, and future ($f$) and past ($p$) are unique:

$$\varphi_l = \forall x, y, y'. \text{right}(x, y) \land \text{right}(x, y') \rightarrow y = y'$$

$$\varphi_f = \forall x, x'. \text{right}(x, y) \land \text{right}(x', y) \rightarrow x = x'$$

$$\varphi_r = \forall x, y, y'. \text{future}(x, y) \land \text{future}(x, y') \rightarrow y = y'$$

$$\varphi_p = \forall x, x'. \text{future}(x, y) \land \text{future}(x', y) \rightarrow x = x'$$

The Impossibility of FO Query Optimisation

Trakhtenbrot’s Theorem has severe consequences for static FO query optimisation

**Theorem 9.2 (Exercise):** All of the following decision problems are undecidable:

- Query equivalence
- Query emptiness
- Query containment

~ “perfect” FO query optimisation is impossible

Other important questions about FO queries are also undecidable, for example:

- Is a given FO query domain independent?
Is Query Optimisation Futile?

Not quite: things are simpler for conjunctive queries

**Example 9.3:** Conjunctive query containment:

<table>
<thead>
<tr>
<th>$Q_1$</th>
<th>$\exists x, y, z. R(x, y) \land R(y, y) \land R(y, z)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$Q_2$</td>
<td>$\exists u, v, w, t. R(u, v) \land R(v, w) \land R(w, t)$</td>
</tr>
</tbody>
</table>

$Q_1$ find $R$-paths of length two with a loop in the middle

$Q_2$ find $R$-paths of length three

$\Rightarrow$ in a loop one can find paths of any length

$\Rightarrow Q_1 \subseteq Q_2$

Summary and Outlook

There are many well-defined static optimisation tasks that are independent of the database

$\Rightarrow$ query equivalence, containment, emptiness

Unfortunately, all of them are undecidable for FO queries

$\Rightarrow$ Slogan: "all interesting questions about FO queries are undecidable"

Open questions:
- More positive results for conjunctive queries
- Measure expressivity rather than just complexity
- Look at query languages beyond first-order logic