TECHNISCHE UNIVERSITÄT DRESDEN

DATABASE THEORY

Lecture 9: Query Optimisation

Markus Krötzsch

Knowledge-Based Systems

TU Dresden, 9th May 2022

Review

We have studied FO queries and the simpler conjunctive queries

Our focus was on query answering complexity:

	Combined complexity	Query complexity	Data complexity
FO queries	PSpace-comp.	PSpace-comp.	in AC^0
Conjunctive queries	NP-comp.	NP-comp.	in AC^0
Tree CQs	in P	in P	in AC^0
Bounded Treewidth CQs	in P	in P	in AC^0
Bounded Hypertree width CQs	in P	in P	in AC^0

Static Query Optimisation

Can we optimise query execution without looking at the database?

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

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., $x \neq x$ or $R(x) \wedge \neg R(x)$)

  $\leadsto$ 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: $Q_1 \sqsubseteq Q_2$)

  $\leadsto$ Generalisation of equivalence:
  $Q_1 \equiv Q_2$ if and only if $Q_1 \sqsubseteq Q_2$ and $Q_2 \sqsubseteq Q_1$

- 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

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)" [Wikidedia article First-order logic]

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)" [Wikidedia 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 φ and a finite model $\mathcal{I}$
- Question: is $\mathcal{I}$ a model for φ, i.e., is φ satisfied in $\mathcal{I}$?
- Corresponds to Boolean query entailment
- PSpace-complete for first-order sentences

Problem 2: Satisfiability Checking

- Given: a logical sentence φ
- Question: does φ 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 φ_1 and φ_2 equivalent?

(2) Are the FO sentences φ_1 and φ_2 equivalent?

$\leadsto$ So FO query equivalence is undecidable?

Back to Query Optimisation

What do these results mean for query optimisation?

Two similar questions:

(1) Are the Boolean FO queries φ_1 and φ_2 equivalent?

(2) Are the FO sentences φ_1 and φ_2 equivalent?

$\leadsto$ So FO query equivalence is undecidable?

However, (1) is not equivalent to (2) but to the following:

(2') Are the FO sentences φ_1 and φ_2 equivalent in all finite interpretations?

$\leadsto$ finite-model reasoning for FO logic

Finite-Model Reasoning

Does it really make a difference?

Finite-Model Reasoning

Does it really make a difference?

Yes. Example formula φ:

$(\forall x.\exists y.R(x,y)) \wedge$	
$(\forall x, y_1, y_2.R(x,y_1) \wedge R(x,y_2) \to y_1 \approx y_2) \wedge$	R is a function ...
$(\forall x_1, x_2, y.R(x_1,y) \wedge R(x_2,y) \to x_1 \approx x_2) \wedge$	... and injective ...
$(\exists y.\forall x.\neg R(x,y))$	... but not surjective

Such a function R can only exist over an infinite domain.
$\leadsto$ over finite models, φ is unsatisfiable
$\leadsto$ φ is finitely equivalent to $\forall x.R(x,x) \wedge \neg R(x,x)$
$\leadsto$ this equivalence does not hold on arbitrary models

Trakhtenbrot's Theorem

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

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.

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

Let's Prove Trakhtenbrot's Theorem

Proof idea: reduce the Halting Problem to finite satisfiability

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

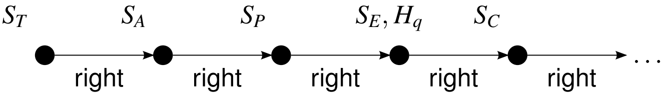
Recall: Turing Machine is given as $\mathcal{M} = \langle Q, q_{\text{start}}, q_{\text{acc}}, \Sigma, \Delta \rangle$
(state set Q, tape alphabet Σ with blank $\textvisiblespace$, transitions $\Delta \subseteq (Q \times \Sigma) \times (Q \times \Sigma \times \{l, r, s\})$)

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

$q \in Q$
$\downarrow$

T	A	P	E	C	O	N	T	E	N	T	S	$\textvisiblespace$	$\cdots$

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



Encoding TM Runs as Relational Structures

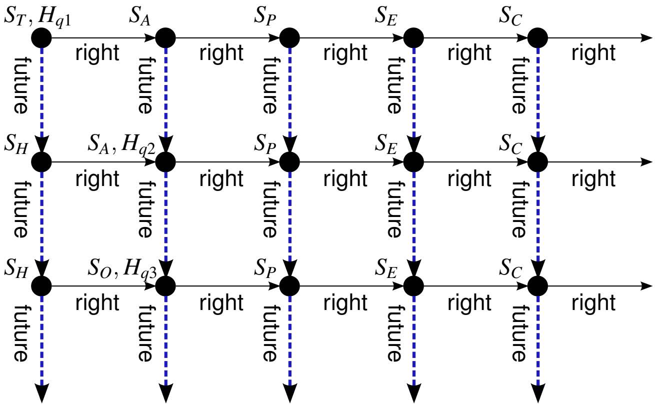
We use several unary predicate symbols to mark tape cells:

- $S_\sigma(\cdot)$ for each $\sigma \in \Sigma$: tape cell contains symbol σ
- $H_q(\cdot)$ 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:

- right$(\cdot, \cdot)$: neighbouring tape cells at same step
- right$^+(\cdot, \cdot)$: transitive super-relation of right
- future$(\cdot, \cdot)$: tape cells at same position in consecutive steps

Intended Database



(right$^+$ is not shown)

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

Defining the Initial Configuration

Require that right$^+$ is a transitive super-relation of right:

$$
\begin{aligned}
\varphi_{\text{right}^+} = {} & \forall x, y.(\text{right}(x, y) \to \text{right}^+(x, y)) \wedge \\
& \forall x, y, z.(\text{right}(x, y) \wedge \text{right}^+(y, z) \to \text{right}^+(x, z))
\end{aligned}
$$

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

$$
\begin{aligned}
\varphi_w = \exists x_1, \ldots, x_n. H_{q_{\text{start}}}(x_1) \wedge \neg\exists z.\text{right}(z, x_1) \wedge {} & \\
& S_{\sigma_1}(x_1) \wedge \neg\exists z.\text{future}(z, x_1) \wedge \text{right}(x_1, x_2) \wedge \\
& S_{\sigma_2}(x_2) \wedge \neg\exists z.\text{future}(z, x_2) \wedge \text{right}(x_2, x_3) \wedge \\
& \ldots \\
& S_{\sigma_n}(x_n) \wedge \neg\exists z.\text{future}(z, x_n) \wedge \\
& \forall y.(\text{right}^+(x_n, y) \to (S_{\llcorner\!\lrcorner}(y) \wedge \neg\exists z.\text{future}(z, y)))
\end{aligned}
$$

$\leadsto$ there can be any number of cells right of the input, but they must contain $\llcorner\!\lrcorner$.

Consistent Tape Contents, Head, and State

A cell can only contain one symbol:

$$\varphi_S = \bigwedge_{\sigma,\sigma'\in\Sigma,\sigma\neq\sigma'} \forall x.(\neg S_\sigma(x) \lor \neg S_{\sigma'}(x))$$

The TM is never at more than one position:

$$\varphi_H = \bigwedge_{q\in Q} \forall x, y. \left(H_q(x) \land \mathsf{right}^+(x, y) \to \bigwedge_{q'\in Q} \neg H_{q'}(y) \right)$$

The TM can only be in one state:

$$\varphi_Q = \bigwedge_{q,q'\in Q,q\neq q'} \forall x.(\neg H_q(x) \lor \neg H_{q'}(x))$$

Transitions

For every non-moving transition $\delta = \langle q, \sigma, q', \sigma', s\rangle \in \Delta$:

$$\varphi_\delta = \forall x. H_q(x) \wedge S_\sigma(x) \to \exists y.\mathsf{future}(x, y) \wedge S_{\sigma'}(y) \wedge H_{q'}(y)$$

For every right-moving transition $\delta = \langle q, \sigma, q', \sigma', r\rangle \in \Delta$:

$$\varphi_\delta = \forall x. H_q(x) \wedge S_\sigma(x) \to \exists y.\mathsf{future}(x, y) \wedge S_{\sigma'}(y) \wedge \exists z.\mathsf{right}(y, z) \wedge H_{q'}(z)$$

For every left-moving transition $\delta = \langle q, \sigma, q', \sigma', l\rangle \in \Delta$:

$$\varphi_\delta = \forall x. H_q(x) \wedge S_\sigma(x) \wedge (\exists v.\mathsf{right}(v, x)) \to \exists y.\mathsf{future}(x, y) \wedge S_{\sigma'}(y) \wedge \exists z.\mathsf{right}(z, y) \wedge H_{q'}(z)$$

Summing all up:

$$\varphi_\Delta = \bigwedge_{\delta \in \Delta} \varphi_\delta$$

Preserve Tape if not Changed by Transition

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

$$\varphi_{\mathsf{mem}} = \forall x, y. \bigwedge_{\sigma\in\Sigma} \left(S_\sigma(x) \wedge \left(\bigwedge_{q\in Q} \neg H_q(x) \right) \wedge \mathsf{future}(x,y) \to S_\sigma(y) \right)$$

Building the Configuration Grid

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

$$\varphi_{fp1} = \forall x_2, y_1.(\exists x_1.\text{right}(x_1, y_1) \wedge \text{future}(x_1, x_2)) \leftrightarrow (\exists y_2.\text{future}(y_1, y_2) \wedge \text{right}(x_2, y_2))$$
$$\varphi_{fp2} = \forall x_1, y_2.(\exists y_1.\text{right}(x_1, y_1) \wedge \text{future}(y_1, y_2)) \leftrightarrow (\exists x_2.\text{future}(x_1, x_2) \wedge \text{right}(x_2, y_2))$$

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

$$\varphi_r = \forall x, y, y'.\text{right}(x, y) \wedge \text{right}(x, y') \rightarrow y \approx y'$$
$$\varphi_l = \forall x, x', y.\text{right}(x, y) \wedge \text{right}(x', y) \rightarrow x \approx x'$$
$$\varphi_f = \forall x, y, y'.\text{future}(x, y) \wedge \text{future}(x, y') \rightarrow y \approx y'$$
$$\varphi_p = \forall x, x', y.\text{future}(x, y) \wedge \text{future}(x', y) \rightarrow x \approx x'$$

Finishing the Proof of Trakhtenbrot's Theorem

We obtain a final FO formula

$$\varphi_{\mathcal{M},w} = \varphi_{\mathsf{right}^+} \wedge \varphi_w \wedge \varphi_S \wedge \varphi_H \wedge \varphi_Q \wedge \varphi_\Delta \wedge \varphi_{\mathsf{mem}} \wedge \varphi_{fp1} \wedge \varphi_{fp2} \wedge \varphi_r \wedge \varphi_l \wedge \varphi_f \wedge \varphi_p$$

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

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

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

$\leadsto$ "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:

$$Q_1 : \quad \exists x, y, z.\ R(x, y) \wedge R(y, y) \wedge R(y, z)$$
$$Q_2 : \quad \exists u, v, w, t.\ R(u, v) \wedge R(v, w) \wedge R(w, t)$$

Q_1 find R-paths of length two with a loop in the middle
Q_2 find R-paths of length three

$\leadsto$ in a loop one can find paths of any length
$\leadsto$ $Q_1 \sqsubseteq Q_2$

Summary and Outlook

There are many well-defined static optimisation tasks that are independent of the database
$\leadsto$ query equivalence, containment, emptiness

Unfortunately, all of them are undecidable for FO queries
$\leadsto$ 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