Database Theory and Knowledge Representation

2nd Lecture

David Carral

University of Montpellier

October 21, 2021

Summary and Outlook

- The relational data model
- Relational queries
- First-order queries

Outline:

- Query expressivity: comparing RA and FO queries
- Complexity of query answering
- Tractable query answering

5. Query Expressivity

Equivalent Queries

The same query can be expressed with different languages:

Example

The query mapping

Who is the director of “The Imitation Game”?

can be expressed using the relational algebra

$$\pi_{Director}(\sigma_{Title=\text{“The Imitation Game”}}(Films))$$

or an FO query

$$\exists y_A.\,\mathrm{Films}(\texttt{"The Imitation Game"}, x_D, y_A)[x_D].$$

How to Compare Query Languages

We have studied two different query languages
$\rightsquigarrow$ how to compare them?

Definition

The set of query mappings that can be described in a query language L is denoted **QM**(L).

- L_1 is subsumed by L_2, written $L_1 \sqsubseteq L_2$, if $\mathbf{QM}(L_1) \subseteq \mathbf{QM}(L_2)$
- L_1 is equivalent to L_2, written $L_1 \equiv L_2$, if $\mathbf{QM}(L_1) = \mathbf{QM}(L_2)$

Theorem

The following query languages are equivalent:

- Relational algebra (RA)
- First-order queries (FO)

Comparing Query Languages: A Simple Example

Example

Consider the RA$^{\setminus\cap}$, which is a restricted version of the RA that only allows for the use of $\{\sigma, \pi, \cup, -, \bowtie, \delta\}$. We can show that RA and RA$^{\setminus\cap}$ are equivalent.

Solution

- Trivial: RA$^{\setminus\cap}$ is subsumed by the RA.
- To show that RA is subsumed by RA$^{\setminus\cap}$ note that, given some RA queries q and s:

$$q \cap s \equiv q \bowtie s$$

RA $\sqsubseteq$ FO

Definition

For a given RA query $q[a_1, \dots, a_n]$, we recursively construct a FO query $\varphi_q[x_{a_1}, \dots, x_{a_n}]$ as follows:

1. if $q = R$ with signature $R[a_1, \dots, a_n]$,[1] then $\varphi_q = R(x_{a_1}, \dots, x_{a_n})$
2. if $n = 1$ and $q = \{\{a_1 \mapsto c\}\}$, then $\varphi_q = (x_{a_1} \approx c)$
3. if $q = \sigma_{a_i = c}(q')$, then $\varphi_q = \varphi_{q'} \land (x_{a_i} \approx c)$
4. if $q = \sigma_{a_i = a_j}(q')$, then $\varphi_q = \varphi_{q'} \land (x_{a_i} \approx x_{a_j})$
5. if $q = \delta_{a_1, \dots, a_n \to b_1, \dots, b_n} q'$,[2] then
   $\varphi_q = \exists x_{a_1}, \dots, x_{a_n}.(\bigwedge_{1 \leq i \leq n} x_{a_i} \approx x_{b_i}) \land \varphi_{q'}[x_{b_1}, \dots, x_{b_n}]$

[1] We assume wlog that all attribute lists in RA expressions respect the global order of attributes.

[2] We assume that $\{a_1, \dots, a_n\} \cap \{b_1, \dots, b_n\} = \emptyset$ without loss of generality.

RA $\sqsubseteq$ FO (cont'd)

Definition (cont'd)

6. if $q = \pi_{a_1,\ldots,a_n}(q')$ for a subquery $q'[b_1,\ldots,b_m]$ with $\{b_1,\ldots,b_m\} = \{a_1,\ldots,a_n\} \cup \{c_1,\ldots,c_k\}$, then $\varphi_q = \exists x_{c_1},\ldots,x_{c_k}.\varphi_{q'}$
7. if $q = q_1 \bowtie q_2$, then $\varphi_q = \varphi_{q_1} \land \varphi_{q_2}$
8. if $q = q_1 \cup q_2$, then $\varphi_q = \varphi_{q_1} \lor \varphi_{q_2}$
9. if $q = q_1 - q_2$, then $\varphi_q = \varphi_{q_1} \land \neg\varphi_{q_2}$

Remarks

- Show that φ_q is equivalent to q via structural induction.
- We have not defined a translation for queries of the form $q \cap s$. Is our proof incomplete?

FO $\sqsubseteq$ RA

To define this direction, we first define a preliminary RA query:

Definition

For a FO query q, a database schema $\mathcal{S}$, and some arbitrary query q; let $Dom^{\mathcal{S}}_{a,q}$ be the following RA expression:

$$\left(\bigcup_{R \in \text{Tables}(\mathcal{S})} \bigcup_{b \in \text{Atts}(R)} \delta_{b \to a}\left(\pi_b(R)\right) \right) \cup \{\, \{\, a \mapsto c \,\} \mid c \in \mathbf{dom}(q) \}.$$

Remark

Note that $Dom^{\mathcal{S}}_{a,q}(\mathcal{I}) = \{\{a \mapsto c\} \mid c \in \mathbf{dom}(\mathcal{I}, q)\}$ for any database $\mathcal{I}$ defined over $\mathcal{S}$.

FO $\sqsubseteq$ RA (CONT'D)

Definition

Consider an FO query $q = \varphi[x_1, \ldots, x_n]$ that is defined for a database with schema $\mathcal{S}$. For every variable x, we use a fresh attribute name a_x.

- if $\varphi = R(t_1, \ldots, t_m)$ with signature $R[a_1, \ldots, a_m]$ with variables $x_1 = t_{v_1}, \ldots, x_n = t_{v_n}$[3] and constants $c_1 = t_{w_1}, \ldots, c_k = t_{w_k}$, then $E_\varphi = \delta_{a_{v_1} \ldots a_{v_n} \to a_{x_1} \ldots a_{x_n}}(\sigma_{a_{w_1}=c_1}(\ldots \sigma_{a_{w_k}=c_k}(R) \ldots))$
- if $\varphi = (x \approx c)$, then $E_\varphi = \{\{a_x \mapsto c\}\}$
- if $\varphi = (x \approx y)$, then $E_\varphi = \sigma_{a_x=a_y}(Dom^{\mathcal{S}}_{a_x,\varphi} \bowtie Dom^{\mathcal{S}}_{a_y,\varphi})$
- other forms of equality atoms are analogous

[3] W.l.o.g., we assume that each of these variables occurs at most once in φ.

FO $\sqsubseteq$ RA (CONT'D)

Definition (cont'd)

- if $\varphi = \neg\psi$, then $E_\varphi = (Dom^{\mathcal{S}}_{a_{x_1},\varphi} \bowtie \ldots \bowtie Dom^{\mathcal{S}}_{a_{x_n},\varphi}) - E_\psi$
- if $\varphi = \varphi_1 \wedge \varphi_2$, then $E_\varphi = E_{\varphi_1} \bowtie E_{\varphi_2}$
- if $\varphi = \exists y.\psi$ where ψ has free variables $y, x_1, \ldots, x_n$, then $E_\varphi = \pi_{a_{x_1},\ldots,a_{x_n}} E_\psi$

Remark

The cases for $\vee$ and $\forall$ can be constructed from the above:

$$E_{\forall y.\psi} \equiv E_{\neg\exists y.\neg\psi} \quad E_{\psi\vee\varphi} \equiv E_{\neg(\neg\psi\wedge\neg\varphi)}$$

6. Complexity of Query Answering

Review: The Relational Calculus

What we have learned so far:

- There are many ways to describe databases:
  $\rightsquigarrow$ set of tables, set of facts, (hyper)graphs
- We have studied two different languages:
  $\rightsquigarrow$ relational algebra and FO queries

The above languages are equivalent: The Relational Calculus

Outlook:

- Next question: How hard is it to answer such queries?
- Related question: Are you familiar with computational complexity theory?

How to Measure Complexity of Queries?

- Complexity classes often for decision problems
  $\leadsto$ database queries return many results
- The size of a query result can be very large
  $\leadsto$ it would not be fair to measure this as “complexity”
- In practice, database instances are much larger than queries
  $\leadsto$ can we take this into account?

Query Answering as Decision Problem

We consider the following decision problems:

- **Boolean Query Entailment:** given a Boolean query q and a database instance $\mathcal{I}$, does $\mathcal{I} \models q$ hold?
- **Query Answering Problem:** given an n-ary query q, a database instance $\mathcal{I}$ and a tuple $\langle c_1, \ldots, c_n \rangle$, does $\langle c_1, \ldots, c_n \rangle \in M[q](\mathcal{I})$ hold?
- **Query Emptiness Problem:** given a query q and a database instance $\mathcal{I}$, does $M[q](\mathcal{I}) \neq \emptyset$ hold?

Discussion

These problems are computationally equivalent.

The Size of the Input

Definition: Combined Complexity

Input: Boolean query q and database instance $\mathcal{I}$
Output: Does $\mathcal{I} \models q$ hold?

$\rightsquigarrow$ “2KB query/2TB database” = “2TB query/2KB database”

Study worst-case complexity of algorithms for fixed queries:

Definition: Data Complexity

Input: database instance $\mathcal{I}$
Output: Does $\mathcal{I} \models q$ hold? (for fixed q)

We can also fix the database and vary the query:

Definition: Query Complexity

Input: Boolean query q
Output: Does $\mathcal{I} \models q$ hold? (for fixed $\mathcal{I}$)

Review: Computation and Complexity Theory

The Turing Machine (1)

Computation is usually modelled with Turing Machines (TMs)
$\rightsquigarrow$ “algorithm” = “something implemented on a TM”

A TM is an automaton with (unlimited) working memory:

- It has a finite set of states Q
- Q includes a start state q_{start} and an accept state q_{acc}
- The memory is a tape with numbered cells $0, 1, 2, \ldots$
- Each tape cell holds one symbol from the set of tape symbols Γ
- There is a special symbol   for empty tape cells
- The TM has a transition relation $\Delta \subseteq (Q \times \Gamma) \times (Q \times \Gamma \times \{l, r, s\})$
- Δ might be a partial function $(Q \times \Gamma) \to (Q \times \Gamma \times \{l, r, s\})$
  $\rightsquigarrow$ deterministic TM (DTM); otherwise nondeterministic TM

There are many different but equivalent ways of defining TMs.

The Turing Machine (2)

TMs operate step-by-step:

- At every moment, the TM is in one state $q \in Q$ with its read/write head at a certain tape position $p \in \mathbb{N}$, and the tape has a certain contents $\sigma_0\sigma_1\sigma_2 \cdots$ with all $\sigma_i \in \Gamma$ $\rightsquigarrow$ current configuration of the TM
- The TM starts in state q_{start} and at tape position 0.
- Transition $\langle q, \sigma, q', \sigma', d\rangle \in \Delta$ means:
  if in state q and the tape symbol at its current position is σ,
  then change to state q', write symbol σ' to tape, move head by d (left/right/stay)
- If there is more than one possible transition, the TM picks one nondeterministically
- The TM halts when there is no possible transition for the current configuration (possibly never)

A computation path (or run) of a TM is a sequence of configurations that can be obtained by some choice of transition.

The Turing Machine (3)

A Turing machine can be described with different levels of precision:

- Formal level: define the states, transition function, alphabet, etc; can be done via diagram (see example in the board).
- Implementational level: describe how the machine works at an implementational level; e.g., describe encodings precisely as well as how the different tapes will be used.
- High level: give an intuitive description of how the Turing machine works.

Example

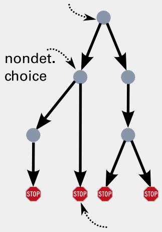
Discuss how to implement a Turing machine that computes the result of the join operator.

Languages Accepted by TMs

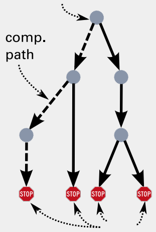
The (nondeterministic) TM accepts an input $\sigma_1 \cdots \sigma_n \in (\Gamma \setminus \{\ \})^*$ if, when started on the tape $\sigma_1 \cdots \sigma_n \ \ \cdots$,

1. the TM halts on every computation path and
2. there is at least one computation path that halts in the accepting state $q_{\text{acc}} \in Q$.

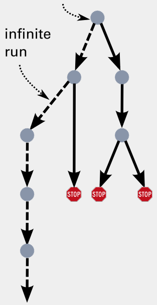
accept:



reject:



reject (not halting):



Solving Computation Problems with TMs

A decision problem is a language $\mathcal{L}$ of words over $\Sigma = \Gamma \setminus \{\ \}$
$\rightsquigarrow$ the set of all inputs for which the answer is "yes"

A TM decides a decision problem $\mathcal{L}$ if it halts on all inputs and accepts exactly the words in $\mathcal{L}$

TMs take time (number of steps):

- $\textsc{Time}(f(n))$: Problems that can be decided by a DTM in $O(f(n))$ steps, where f is a function of the input length n
- $\textsc{NTime}(f(n))$: Problems that can be decided by a TM in at most $O(f(n))$ steps **on any of its computation paths**

We can also consider space (number of cells) as a restriction.

Reminder

Given some functions f and g defined over the natural numbers, we write $f(x) = O(g(x))$ to indicate that there are some $n, x_0 > 0$ such that $|f(x)| \leq ng(x)$ for all $x \geq x_0$.

Some Common Complexity Classes

$$\mathrm{P} = \mathrm{PT{\scriptsize IME}} = \bigcup_{k \geq 1} \mathrm{T{\scriptsize IME}}(n^k) \qquad \mathrm{NP} = \bigcup_{k \geq 1} \mathrm{NT{\scriptsize IME}}(n^k)$$

$$\mathrm{E{\scriptsize XP}} = \mathrm{E{\scriptsize XP}T{\scriptsize IME}} = \bigcup_{k \geq 1} \mathrm{T{\scriptsize IME}}(2^{n^k}) \qquad \mathrm{NE{\scriptsize XP}} = \mathrm{NE{\scriptsize XP}T{\scriptsize IME}} = \bigcup_{k \geq 1} \mathrm{NT{\scriptsize IME}}(2^{n^k})$$

$$\mathrm{2E{\scriptsize XP}} = \mathrm{2E{\scriptsize XP}T{\scriptsize IME}} = \bigcup_{k \geq 1} \mathrm{T{\scriptsize IME}}(2^{2^{n^k}}) \qquad \mathrm{N2E{\scriptsize XP}} = \mathrm{N2E{\scriptsize XP}T{\scriptsize IME}} = \bigcup_{k \geq 1} \mathrm{NT{\scriptsize IME}}(2^{2^{n^k}})$$

$$\mathrm{ET{\scriptsize IME}} = \bigcup_{k \geq 1} \mathrm{T{\scriptsize IME}}(2^{nk})$$

$$\mathrm{L} = \mathrm{L{\scriptsize OG}S{\scriptsize PACE}} = \mathrm{S{\scriptsize PACE}}(\log n) \qquad \mathrm{NL} = \mathrm{NL{\scriptsize OG}S{\scriptsize PACE}} = \mathrm{NS{\scriptsize PACE}}(\log n)$$

$$\mathrm{PS{\scriptsize PACE}} = \bigcup_{k \geq 1} \mathrm{S{\scriptsize PACE}}(n^k)$$

$$\mathrm{E{\scriptsize XP}S{\scriptsize PACE}} = \bigcup_{k \geq 1} \mathrm{S{\scriptsize PACE}}(2^{n^k})$$

How to Measure Query Answering Complexity

Query answering as decision problem
$\rightsquigarrow$ 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:

$$\mathrm{P} \subseteq \mathrm{NP} \subseteq \textsc{PSpace} \subseteq \textsc{ExpTime}$$

7. Evaluating FO Queries

An Algorithm for Evaluating FO Queries

function Eval(φ, $\mathcal{I}$)

```
switch ($\varphi$) {
    case $p(c_1, \ldots, c_n)$ : return $p(c_1, \ldots, c_n) \in \mathcal{I}$
    case $\neg\psi$ : return $\neg$Eval($\psi$, $\mathcal{I}$)
    case $\psi_1 \wedge \psi_2$ : return Eval($\psi_1$, $\mathcal{I}$) $\wedge$ Eval($\psi_2$, $\mathcal{I}$)
    case $\exists x.\psi$ :
        for $c \in \Delta^{\mathcal{I}}$ {
            if Eval($\psi[x \mapsto c]$, $\mathcal{I}$) then return true
        }
        return false
}
```

Remark

The formula φ is a Boolean FO query. How can we extend the above procedure to solve query answering?

FO Algorithm Worst-Case Runtime

Let m be the size of φ, and let $n = |\mathcal{I}|$ (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?
  $\leadsto$ $|\Delta^{\mathcal{I}}| \leq n$ per recursion level
  $\leadsto$ at most n^m iterations
- Checking $P(c_1, \ldots, c_n) \in \mathcal{I}$ 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 φ, and let $n = |\mathcal{I}|$ (total table sizes)

Runtime in $m \cdot n^{m+1}$

Theorem

Time complexity of FO query evaluation

- Combined complexity: in $\textsc{ExpTime}$
- Data complexity (m is constant): in P
- Query complexity (n is constant): in $\textsc{ExpTime}$

FO Algorithm Worst-Case Memory Usage

We can get better complexity bounds by looking at memory!

Let m be the size of φ, and let $n = |\mathcal{I}|$ (total table sizes)

$\leadsto$ on the whiteboard

Theorem

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

Remark

One can show that FO query entailment is $\textsc{PSpace}$-hard via reduction to True QBF.

Summary

Summary and Outlook

We have covered the following topics:

- RA $\equiv$ FO
- The relational calculus
- Time complexity of query entailment over FO queries

Future Content:

- Space complexity of query entailment over FO queries
- Tractable query entailment