TECHNISCHE UNIVERSITÄT DRESDEN

DATABASE THEORY

Lecture 2: First-Order Queries

Markus Krötzsch
Knowledge-Based Systems

TU Dresden, 9th Apr 2019

What is a Query?

The relational queries considered so far produced a result table from a database. Other query languages can be completely different, but they usually agree on this:

Definition 2.1:

- Syntax: a query expression q is a word from a query language (algebra expression, logical expression, etc.)
- Semantics: a query mapping $M[q]$ is a function that maps a database instance $\mathcal{I}$ to a database table $M[q](\mathcal{I})$

$\leadsto$ for some semantics, query mappings are not defined on all database instances

Generic Queries

We only consider queries that do not depend on the concrete names given to constants in the database:

Definition 2.2: A query q is generic if, for every bijective renaming function μ : **dom** $\to$ **dom** and database instance $\mathcal{I}$:

$$\mu(M[q](\mathcal{I})) = M[\mu(q)](\mu(\mathcal{I})).$$

In this case, $M[q]$ is closed under isomorphisms.

Review: Example from Previous Lecture

Lines:

Line	Type
85	bus
3	tram
F1	ferry
...	...

Stops:

SID	Stop	Accessible
17	Hauptbahnhof	true
42	Helmholtzstr.	true
57	Stadtgutstr.	true
123	Gustav-Freytag-Str.	false
...	...	...

Connect:

From	To	Line
57	42	85
17	789	3
...	...	...

Every table has a schema:

- Lines[Line:string, Type:string]
- Stops[SID:int, Stop:string, Accessible:bool]
- Connect[From:int, To:int, Line:string]

First-order Logic as a Query Language

Idea: database instances are finite first-order interpretations
⇝ use first-order formulae as query language
⇝ use unnamed perspective (more natural here)

Examples (using schema as in previous lecture):

- Find all bus lines: $\text{Lines}(x, \texttt{"bus"})$
- Find all possible types of lines: $\exists y.\text{Lines}(y, x)$
- Find all lines that depart from an accessible stop:

$$\exists y_{\text{SID}}, y_{\text{Stop}}, y_{\text{To}}.(\text{Stops}(y_{\text{SID}}, y_{\text{Stop}}, \texttt{"true"}) \wedge \text{Connect}(y_{\text{SID}}, y_{\text{To}}, x_{\text{Line}}))$$

First-order Logic with Equality: Syntax

Basic building blocks:

- Predicate names with an arity ≥ 0: p, q, Lines, Stops
- Variables: x, y, z
- Constants: a, b, c
- Terms are variables or constants: s, t

Formulae of first-order logic are defined as usual:

$$\varphi ::= p(t_1, \ldots, t_n) \mid t_1 \approx t_2 \mid \neg\varphi \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \exists x.\varphi \mid \forall x.\varphi$$

where p is an n-ary predicate, t_i are terms, and x is a variable.

- An atom is a formula of the form $p(t_1, \ldots, t_n)$
- A literal is an atom or a negated atom
- Occurrences of variables in the scope of a quantifier are bound; other occurrences of variables are free

First-order Logic Syntax: Simplifications

We use the usual shortcuts and simplifications:

- flat conjunctions ($\varphi_1 \wedge \varphi_2 \wedge \varphi_3$ instead of $(\varphi_1 \wedge (\varphi_2 \wedge \varphi_3))$)
- flat disjunctions (similar)
- flat quantifiers ($\exists x, y, z.\varphi$ instead of $\exists x.\exists y.\exists z.\varphi$)
- $\varphi \to \psi$ as shortcut for $\neg\varphi \vee \psi$
- $\varphi \leftrightarrow \psi$ as shortcut for $(\varphi \to \psi) \wedge (\psi \to \varphi)$
- $t_1 \not\approx t_2$ as shortcut for $\neg(t_1 \approx t_2)$

But we always use parentheses to clarify nesting of $\wedge$ and $\vee$:

No "$\varphi_1 \wedge \varphi_2 \vee \varphi_3$"!

First-order Logic with Equality: Semantics

First-order formulae are evaluated over interpretations $\langle \Delta^{\mathcal{I}}, \cdot^{\mathcal{I}} \rangle$, where $\Delta^{\mathcal{I}}$ is the domain.
To interpret formulas with free variables, we need a variable assignment $\mathcal{Z} : \text{Var} \to \Delta^{\mathcal{I}}$.

- constants a interpreted as $a^{\mathcal{I},\mathcal{Z}} = a^{\mathcal{I}} \in \Delta^{\mathcal{I}}$
- variables x interpreted as $x^{\mathcal{I},\mathcal{Z}} = \mathcal{Z}(x) \in \Delta^{\mathcal{I}}$
- n-ary predicates p interpreted as $p^{\mathcal{I}} \subseteq (\Delta^{\mathcal{I}})^n$

A formula φ can be satisfied by $\mathcal{I}$ and $\mathcal{Z}$, written $\mathcal{I}, \mathcal{Z} \models \varphi$:

- $\mathcal{I}, \mathcal{Z} \models p(t_1, \ldots, t_n)$ if $\langle t_1^{\mathcal{I},\mathcal{Z}}, \ldots, t_n^{\mathcal{I},\mathcal{Z}} \rangle \in p^{\mathcal{I}}$
- $\mathcal{I}, \mathcal{Z} \models t_1 \approx t_2$ if $t_1^{\mathcal{I},\mathcal{Z}} = t_2^{\mathcal{I},\mathcal{Z}}$
- $\mathcal{I}, \mathcal{Z} \models \neg\varphi$ if $\mathcal{I}, \mathcal{Z} \not\models \varphi$
- $\mathcal{I}, \mathcal{Z} \models \varphi \wedge \psi$ if $\mathcal{I}, \mathcal{Z} \models \varphi$ and $\mathcal{I}, \mathcal{Z} \models \psi$
- $\mathcal{I}, \mathcal{Z} \models \varphi \vee \psi$ if $\mathcal{I}, \mathcal{Z} \models \varphi$ or $\mathcal{I}, \mathcal{Z} \models \psi$
- $\mathcal{I}, \mathcal{Z} \models \exists x.\varphi$ if there is $\delta \in \Delta^{\mathcal{I}}$ with $\mathcal{I}, \{x \mapsto \delta\}, \mathcal{Z} \models \varphi$
- $\mathcal{I}, \mathcal{Z} \models \forall x.\varphi$ if for all $\delta \in \Delta^{\mathcal{I}}$ we have $\mathcal{I}, \{x \mapsto \delta\}, \mathcal{Z} \models \varphi$

First-order Logic Queries

Definition 2.3: An n-ary first-order query q is an expression $\varphi[x_1, \ldots, x_n]$ where $x_1, \ldots, x_n$ are exactly the free variables of φ (in a specific order).

Definition 2.4: An answer to $q = \varphi[x_1, \ldots, x_n]$ over an interpretation $\mathcal{I}$ is a tuple $\langle a_1, \ldots, a_n \rangle$ of constants such that

$$\mathcal{I} \models \varphi[x_1/a_1, \ldots, x_n/a_n]$$

where $\varphi[x_1/a_1, \ldots, x_n/a_n]$ is φ with each free x_i replaced by a_i.

The result of q over $\mathcal{I}$ is the set of all answers of q over $\mathcal{I}$.

Boolean Queries

A Boolean query is a query of arity 0
$\leadsto$ we simply write φ instead of $\varphi[]$
$\leadsto$ φ is a closed formula (a.k.a. sentence)

What does a Boolean query return?

Two possible cases:

- $\mathcal{I} \not\models \varphi$, then the result of φ over $\mathcal{I}$ is $\emptyset$ (the empty table)
- $\mathcal{I} \models \varphi$, then the result of φ over $\mathcal{I}$ is $\{\langle\rangle\}$ (the unit table)

Interpreted as Boolean check with result true or false (match or no match)

Domain Dependence

We have defined FO queries over interpretations
$\leadsto$ How exactly do we get from databases to interpretations?

- Constants are just interpreted as themselves: $a^{\mathcal{I}} = a$
- Predicates are interpreted according to the table contents
- But what is the domain of the interpretation?

Example 2.5: What should the following queries return?

(1) $\neg\text{Lines}(x, \texttt{"bus"})[x]$

(2) $(\text{Connect}(x_1, \texttt{"42"}, \texttt{"85"}) \vee \text{Connect}(\texttt{"57"}, x_2, \texttt{"85"}))[x_1, x_2]$

(3) $\forall y.p(x, y)[x]$

$\leadsto$ Answers depend on the interpretation domain, not just on the database contents

Natural Domain

First possible solution: the natural domain

Natural domain semantics (ND):

- fix the interpretation domain to **dom** (infinite)
- query answers might be infinite (not a valid result table)
  $\leadsto$ query result undefined for such databases

Natural Domain: Examples

Query answers under natural domain semantics:

(1) $\neg\text{Lines}(x, \texttt{"bus"})[x]$
Undefined on all databases

(2) $(\text{Connect}(x_1, \texttt{"42"}, \texttt{"85"}) \vee \text{Connect}(\texttt{"57"}, x_2, \texttt{"85"}))[x_1, x_2]$
Undefined on databases with matching x_1 or x_2 in Connect, otherwise empty

(3) $\forall y.p(x, y)[x]$
Empty on all databases

Active Domain

Alternative: restrict to constants that are really used
$\rightsquigarrow$ active domain

- for a database instance $\mathcal{I}$, **adom**$(\mathcal{I})$ is the set of constants used in relations of $\mathcal{I}$
- for a query q, **adom**(q) is the set of constants in q
- **adom**$(\mathcal{I}, q)$ = **adom**$(\mathcal{I}) \cup$ **adom**(q)

Active domain semantics (AD):

consider database instance as interpretation over **adom**$(\mathcal{I}, q)$

Active Domain: Examples

Query answers under active domain semantics:

(1) $\neg\text{Lines}(x, \texttt{"bus"})[x]$
Let $q' = \text{Lines}(x, \texttt{"bus"})[x]$. The answer is **adom**$(\mathcal{I}, q) \setminus M[q'](\mathcal{I})$

(2) $(\underbrace{\text{Connect}(x_1, \texttt{"42"}, \texttt{"85"})}_{\varphi_1[x_1]} \vee \underbrace{\text{Connect}(\texttt{"57"}, x_2, \texttt{"85"})}_{\varphi_2[x_2]})[x_1, x_2]$
The answer is $M[\varphi_1](\mathcal{I}) \times$ **adom**$(\mathcal{I}, q) \cup$ **adom**$(\mathcal{I}, q) \times M[\varphi_2](\mathcal{I})$

(3) $\forall y.p(x, y)[x] \rightsquigarrow$ see board

Domain Independence

Observation: some queries do not depend on the domain

- $\text{Stops}(x, y, \texttt{"true"})[x, y]$
- $(x \approx a)[x]$
- $p(x) \wedge \neg q(x)[x]$
- $r(x) \wedge \forall y.(q(x, y) \to p(x, y))[x]$ (exercise: why?)

In contrast, all example queries on the previous few slides are not domain independent

Domain independent semantics (DI):

consider only domain independent queries
use any domain **adom**$(\mathcal{I}, q) \subseteq \Delta^{\mathcal{I}} \subseteq$ **dom** for interpretation

How to Compare Query Languages

We have seen three ways of defining FO query semantics
$\leadsto$ how to compare them?

Definition 2.6: 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 $\mathsf{L}_1 \sqsubseteq \mathsf{L}_2$, if $\mathbf{QM}(\mathsf{L}_1) \subseteq \mathbf{QM}(\mathsf{L}_2)$
- L_1 is equivalent to L_2, written $\mathsf{L}_1 \equiv \mathsf{L}_2$, if $\mathbf{QM}(\mathsf{L}_1) = \mathbf{QM}(\mathsf{L}_2)$

We will also compare query languages under named perspective with query languages under unnamed perspective.
This is possible since there is an easy one-to-one correspondence between query mappings of either kind (see exercise).

Equivalence of Relational Query Languages

Theorem 2.7: The following query languages are equivalent:

- Relational algebra RA
- FO queries under active domain semantics AD
- Domain independent FO queries DI

This holds under named and under unnamed perspective.

To prove it, we will show:

$$\mathsf{RA}_{\text{named}} \sqsubseteq \mathsf{DI}_{\text{unnamed}} \sqsubseteq \mathsf{AD}_{\text{unnamed}} \sqsubseteq \mathsf{RA}_{\text{named}}$$

$\mathsf{RA}_{\text{named}} \sqsubseteq \mathsf{DI}_{\text{unnamed}}$

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

We assume without loss of generality that all attribute lists in RA expressions respect the global order of attributes.

- if $q = R$ with signature $R[a_1, \ldots, a_n]$, then $\varphi_q = R(x_{a_1}, \ldots, x_{a_n})$
- if $n = 1$ and $q = \{\{a_1 \mapsto c\}\}$, then $\varphi_q = (x_{a_1} \approx c)$
- if $q = \sigma_{a_i = c}(q')$, then $\varphi_q = \varphi_{q'} \wedge (x_{a_i} \approx c)$
- if $q = \sigma_{a_i = a_j}(q')$, then $\varphi_q = \varphi_{q'} \wedge (x_{a_i} \approx x_{a_j})$
- if $q = \delta_{b_1, \ldots, b_n \to a_1, \ldots, a_n} q'$, then
  $\varphi_q = \exists y_{b_1}, \ldots, y_{b_n}.(x_{a_1} \approx y_{b_1}) \wedge \ldots \wedge (x_{a_n} \approx y_{b_n}) \wedge \varphi_{q'}[y_{B_1}, \ldots, y_{B_n}]$
  (Here we assume that the $a_1, \ldots, a_n$ in $\delta_{b_1, \ldots, b_n \to a_1, \ldots, a_n}$ are written in the order of attributes, while $b_1, \ldots, b_n$ might be in another order. We use $\{B_1, \ldots, B_n\} = \{b_1, \ldots, b_n\}$ to denote the ordered version of the b_i attributes. $\varphi_{q'}[y_{B_1}, \ldots, y_{B_n}]$ is like $\varphi_{q'}$ but using variables y_{B_i}.)

$\mathsf{RA}_{\text{named}} \sqsubseteq \mathsf{DI}_{\text{unnamed}}$ (cont'd)

Remaining cases:

- 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 $\varphi_q = \exists x_{c_1}, \ldots, x_{c_k}.\varphi_{q'}$
- if $q = q_1 \bowtie q_2$ then $\varphi_q = \varphi_{q_1} \wedge \varphi_{q_2}$
- if $q = q_1 \cup q_2$ then $\varphi_q = \varphi_{q_1} \vee \varphi_{q_2}$
- if $q = q_1 - q_2$ then $\varphi_q = \varphi_{q_1} \wedge \neg\varphi_{q_2}$

One can show that $\varphi_q[x_{a_1}, \ldots, x_{a_n}]$ is domain independent and equivalent to q
$\leadsto$ exercise

$\mathsf{DI}_{\text{unnamed}} \sqsubseteq \mathsf{AD}_{\text{unnamed}}$

This is easy to see:

- Consider an FO query q that is domain independent
- The semantics of q is the same for any domain **adom** $\subseteq \Delta^{\mathcal{I}} \subseteq$ **dom**
- In particular, the semantics of q is the same under active domain semantics
- Hence, for every DI query, there is an equivalent AD query

$\mathsf{AD}_{\text{unnamed}} \sqsubseteq \mathsf{RA}_{\text{named}}$

Consider an AD query $q = \varphi[x_1, \ldots, x_n]$.

For an arbitrary attribute name a, we can construct an RA expression $E_{a,\textbf{adom}}$ such that $E_{a,\textbf{adom}}(\mathcal{I}) = \{\{a \mapsto c\} \mid c \in \textbf{adom}(\mathcal{I}, q)\}$
$\leadsto$ exercise

For every variable x, we use a distinct attribute name a_x

- if $\varphi = R(t_1, \ldots, t_m)$ with signature $R[a_1, \ldots, a_m]$ with variables $x_1 = t_{v_1}, \ldots, x_n = t_{v_n}$ 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 $E_\varphi = \sigma_{a_x=a_y}(E_{a_x,\textbf{adom}} \bowtie E_{a_y,\textbf{adom}})$
- other forms of equality atoms are similar

$\mathsf{AD}_{\text{unnamed}} \sqsubseteq \mathsf{RA}_{\text{named}}$ (cont'd)

Remaining cases:

- if $\varphi = \neg\psi$, then $E_\varphi = (E_{a_{x_1},\textbf{adom}} \bowtie \ldots \bowtie E_{a_{x_n},\textbf{adom}}) - E_\psi$
- 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 $E_\varphi = \pi_{a_{x_1},\ldots,a_{x_n}} E_\psi$

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

A note on order: The translation yields an expression $E_\varphi[a_{x_1}, \ldots, a_{x_n}]$. For this to be equivalent to the query $\varphi[x_1, \ldots, x_n]$, we must choose the attribute names such that their global order is $a_{x_1}, \ldots, a_{x_n}$. This is clearly possible, since the names are arbitrary and we have infinitely many names available.

How to find DI queries?

Domain independent queries are arguably most intuitive, since their result does not depend on special assumptions.

$\leadsto$ How can we check if a query is in DI? Unfortunately, we can't:

Theorem 2.8: Given a FO query q, it is undecidable if $q \in$ DI.

$\leadsto$ find decidable sufficient conditions for a query to be in DI

A Normal Form for Queries

We first define a normal form for FO queries:
Safe-Range Normal Form (SRNF)

- Rename variables apart (distinct quantifiers bind distinct variables, bound variables distinct from free variables)
- Eliminate all universal quantifiers: $\forall y.\psi \mapsto \neg\exists y.\neg\psi$
- Push negations inwards:
  - $\neg(\varphi \wedge \psi) \mapsto (\neg\varphi \vee \neg\psi)$
  - $\neg(\varphi \vee \psi) \mapsto (\neg\varphi \wedge \neg\psi)$
  - $\neg\neg\psi \mapsto \psi$

Safe-Range Queries

Let φ be a formula in SRNF. The set $\mathsf{rr}(\varphi)$ of range-restricted variables of φ is defined recursively:

$$\mathsf{rr}(R(t_1,\ldots,t_n)) = \{x \mid x \text{ a variable among the } t_1,\ldots,t_n\}$$
$$\mathsf{rr}(x \approx a) = \{x\}$$
$$\mathsf{rr}(x \approx y) = \emptyset$$
$$\mathsf{rr}(\varphi_1 \wedge \varphi_2) = \begin{cases} \mathsf{rr}(\varphi_1) \cup \{x,y\} & \text{if } \varphi_2 = (x \approx y) \text{ and } \{x,y\} \cap \mathsf{rr}(\varphi_1) \neq \emptyset \\ \mathsf{rr}(\varphi_1) \cup \mathsf{rr}(\varphi_2) & \text{otherwise} \end{cases}$$
$$\mathsf{rr}(\varphi_1 \vee \varphi_2) = \mathsf{rr}(\varphi_1) \cap \mathsf{rr}(\varphi_2)$$
$$\mathsf{rr}(\exists y.\psi) = \begin{cases} \mathsf{rr}(\psi) \setminus \{y\} & \text{if } y \in \mathsf{rr}(\psi) \\ \textbf{throw new } \text{NotSafeException()} & \text{if } y \notin \mathsf{rr}(\psi) \end{cases}$$
$$\mathsf{rr}(\neg\psi) = \emptyset \quad \text{if } \mathsf{rr}(\psi) \text{ is defined (no exception)}$$

Safe-Range Queries

Definition 2.9: An FO query $q = \varphi[x_1,\ldots,x_n]$ is a safe-range query if

$$\mathsf{rr}(\mathsf{SRNF}(\varphi)) = \{x_1,\ldots,x_n\}.$$

Safe-range queries are domain independent.
One can show a much stronger result:

Theorem 2.10: The following query languages are equivalent:

- Safe-range queries SR
- Relational algebra RA
- FO queries under active domain semantics AD
- Domain independent FO queries DI

Tuple-Relational Calculus

There are more equivalent ways to define a relational query language

Example: Codd's tuple calculus

- Based on named perspective
- Use first-order logic, but variables range over sorted tuples (rows) instead of values
- Use expressions like x : From,To,Line to declare sorts of variables in queries
- Use expressions like x.From to access a specific value of a tuple
- Example: Find all lines that depart from an accessible stop

$$\{x : \text{Line} \mid \exists y : \text{SID,Stop,Accessible}.(\text{Stops}(y) \wedge y.\text{Accessible} \approx \texttt{"true"}$$
$$\wedge \exists z : \text{From,To,Line}.(\text{Connect}(z) \wedge z.\text{From} \approx y.\text{SID}$$
$$\wedge z.\text{Line} \approx x.\text{Line}))\}$$

Summary and Outlook

First-order logic gives rise to a relational query language

The problem of domain dependence can be solved in several ways

All common definitions lead to equivalent calculi
$\leadsto$ "relational calculus"

Open questions:

- How hard is it to actually answer such queries? (next lecture)
- How can we study the expressiveness of query languages?
- Are there interesting query languages that are not equivalent to RA?