Ontology and Database Systems: Foundations of Database Systems

Part 3: First-order Query Languages

Werner Nutt

Faculty of Computer Science
European Master in Computatonal Logic

A.Y. 2015/2016

unibz
Freie Universität Bozen
Libera Università di Bolzano
Università Liedia de Bulsan

How Are Algebra and Calculus Related?

We have seen several queries that could be expressed both in

- Relational Calculus
- Relational Algebra.

Are both languages the same? Or are they different?

How can we formally express hypotheses about their relationship?

How could we prove that they are the same? And how that they are different?

unibz

Relationship between Algebra and Calculus

Theorem

For every Relational Algebra expression E one can compute in polynomial time a first-order formula ϕ such that

$$E(\mathbf{I}) = Q_\phi(\mathbf{I})$$

for all instances $\mathbf{I}$

Proof.

Induction over the structure of algebra expressions. See Lemma 5.3.1 in [AHV95]. □

If the algebra expression E contains comparisons in the selection and join conditions, then ϕ will have comparisons

What about the converse statement?

unibz

Safe Queries

Proposition

For every algebra expression E and every instance $\mathbf{I}$, the set $E(\mathbf{I})$ is finite

Proof.

How? □

Definition

Let Q_ϕ be a calculus query.
We say that Q_ϕ is *safe* if $Q_\phi(\mathbf{I})$ is finite for all instances $\mathbf{I}$.

So, all algebra queries are safe. What about calculus queries?

unibz

Negation and Safety

Consider

$$Q = \{(i, n, f) \mid \neg \texttt{Course}(i, n, f)\}$$

What is $Q(\mathbf{I}_{\text{univ}})$?

Can we (automatically) tell safe queries from unsafe queries?

Different Kinds of Satisfiability

Recall: Sentences are formulas where all variables are bound (closed formulas)

Definition

Let ϕ be a logical sentence. Then

- ϕ is **first-order satisfiable** (or just satisfiable) if
  there is a first order interpretation $\mathbf{I}$ such that $\mathbf{I} \models \phi$;
- ϕ is **finitely satisfiable** if
  there is a finite first order interpretation $\mathbf{I}$ such that $\mathbf{I} \models \phi$;
- ϕ is **database satisfiable** if
  there is a database instance $\mathbf{I}$ such that $\mathbf{I} \models \phi$.

Analogously, we define first-order validity, finite validity, and database validity.

What do you know about the decidability of these properties?

unibz

Undecidabilty of Validity in First-order Logic

Theorem (Church (1) and Trakhtenbrot (2))

1. First-order validity of sentences is undecidable.
2. Finite validity of first-order sentences is undecidable.

Proof.

For both proofs, see the notes by Stephen G. Simpson from Penn State University. Simpson's proof uses formulas with one constant 0, two function symbols, and a binary relation symbol. Note that, if we have equality, we can encode n-ary function symbols into $(n+1)$-ary relation symbols, using functionality axioms for the new symbols. Moreover, equality can be encoded into a new binary relation symbol, using congruence axioms. Thus, the claim holds also for sentences without function symbols and without equality. □

What does this imply for (un)satisfiability of sentences?
. . . and what for sentences valid in all database instances?

unibz

Finite Satisfiability vs. Database Satisfiability

Lemma (Relativization)

For every first-order sentence ϕ we can construct (in polynomial time) a first-order sentence $\tilde{\phi}$ such that the following are equivalent:

- ϕ has a finite model;
- $\tilde{\phi}$ has a model that is a database instance.

Proof (Sketch).

□

Finite Satisfiability vs. Database Satisfiability

Lemma (Relativization)

For every first-order sentence ϕ we can construct (in polynomial time) a first-order sentence $\tilde{\phi}$ such that the following are equivalent:

- ϕ has a finite model;
- $\tilde{\phi}$ has a model that is a database instance.

Proof (Sketch).

Idea: Encode the domain implicitly into $\tilde{\phi}$, using a new relation symbol D.

Translate atoms as $\widetilde{R(s_1, \ldots, s_n)} := R(s_1, \ldots, s_n) \wedge D(s_1) \wedge \cdots \wedge D(s_n)$.

Translate conjunctions as $\widetilde{\psi_1 \wedge \psi_2} := \tilde{\psi}_1 \wedge \tilde{\psi}_1$.

Translate existential quantification as $\widetilde{\exists x\, \psi} := \exists x\, (D(x) \wedge \psi)$.

Translate negation as $\widetilde{\neg\psi} := D(x_1) \wedge \cdots \wedge D(x_n) \wedge \neg\psi$,
where $x_1, \ldots, x_n$ are the free variables of ψ. □

unibz

Finite Satisfiability vs. Database Satisfiability

Lemma (Relativization)

For every first-order sentence ϕ we can construct (in polynomial time) a first-order sentence $\tilde{\phi}$ such that the following are equivalent:

- ϕ has a finite model;
- $\tilde{\phi}$ has a model that is a database instance.

Proof (Sketch).

Idea: Encode the domain implicitly into $\tilde{\phi}$, using a new relation symbol D.

Translate atoms as $\widetilde{R(s_1, \ldots, s_n)} := R(s_1, \ldots, s_n) \wedge D(s_1) \wedge \cdots \wedge D(s_n)$.

Translate conjunctions as $\widetilde{\psi_1 \wedge \psi_2} := \tilde{\psi}_1 \wedge \tilde{\psi}_1$.

Translate existential quantification as $\widetilde{\exists x\, \psi} := \exists x\, (D(x) \wedge \psi)$.

Translate negation as $\widetilde{\neg\psi} := D(x_1) \wedge \cdots \wedge D(x_n) \wedge \neg\psi$,
where $x_1, \ldots, x_n$ are the free variables of ψ. □

What does this tell us about the decidability of database satisfiability?
. . . and what about the decidability of safety?

unibz

Decidability of Safety

Theorem

Safety of Relational Calculus queries is undecidable.

Proof.

□

Decidability of Safety

Theorem

Safety of Relational Calculus queries is undecidable.

Proof.

Encode the finite satisfiability problem using the relativization lemma. Then conclude the claim from Trakhtenbrot's Theorem.

More precisely, for every sentence ϕ, we construct a formula $\psi_\phi(x)$ with free variable x such that the query

$$Q_\phi = \{x \mid \psi_\phi(x)\}$$

is safe if and only if ϕ is unsatisfiable.

Let R be a fresh relation symbol not occurring in our first-order sentences. Then let $\psi_\phi(x) = \neg R(x) \wedge \phi$. Clearly, Q_ϕ is unsatisfiable (and thus safe) iff ϕ is unsatisfiable, and unsafe if ϕ is satisfiable.

□

More Properties of Queries

Definition

Let Q, Q_1, Q_2 be relational calculus queries. We say that

- Q is **satisfiable** iff there is an instance $\mathbf{I}$ such that $Q(\mathbf{I}) \neq \emptyset$ (otherwise, Q is **unsatisfiable**)
- Q_1 and Q_2 are **equivalent** (written $Q_1 \equiv Q_2$) iff $Q_1(\mathbf{I}) = Q_2(\mathbf{I})$ for all instances $\mathbf{I}$
- Q_1 is **contained** in Q_2 (written $Q_1 \sqsubseteq Q_2$) iff $Q_1(\mathbf{I}) \subseteq Q_2(\mathbf{I})$ for all instances $\mathbf{I}$

Can we conclude one property from another one?

If so, which from which, and how?

How can we check these properties?

unibz

More Properties of Queries

Theorem

Satisfiability, equivalence, and containment are undecidable for RelCalc queries

Proof.

Undecidability of satisfiability follows from Trakhtenbrot's theorem, using the relativization lemma. The other two claims can then be shown by reduction. Exercise! □

Back to our original question:

Can all safe queries be expressed in relational algebra?

unibz

What is the Role of the Domain in Query Answering?

- Consider the query

$$Q = \{x \mid \texttt{Person}(x) \wedge \forall\, y\, \texttt{Loves}(x, y)\}.$$

  Is Q safe?

- Consider the instance

$$\mathbf{I}^{\text{db}}_{\text{pers}} = \{\texttt{Person}(\text{Fred}),\ \texttt{Person}(\text{Mary}),\\ \texttt{Loves}(\text{Fred},\ \text{Fred}),\ \texttt{Loves}(\text{Fred},\ \text{Mary})\ \}.$$

  What is $Q(\mathbf{I}^{\text{db}}_{\text{pers}})$?

- Consider the finite interpretation $\mathbf{I}^{\text{fin}}_{\text{pers}}$
  - where the domain is $\{\text{Fred}, \text{Mary}\}$ and
  - where the interpretation of each relation is the same as in $\mathbf{I}^{\text{db}}_{\text{pers}}$.

  What is $Q(\mathbf{I}^{\text{fin}}_{\text{pers}})$?

Could Q possibly be defined in relational algebra?

unibz

Domain Independence

A query where the answer depends

- not only on the interpretation of the relations,
- but also on the domain

is **domain dependent**.

All other queries are **domain independent**.

Can you give an example of a relational algebra query that is domain dependent?

We make this more formal on the following slides

unibz

Active Domain

When introducing the semantics of calculus queries, we defined the domain of the interpretation $\mathbf{I}$ as

$$\Delta^{\mathbf{I}} = \mathbf{dom}.$$

However, there are more options.

For an instance $\mathbf{I}$ and a query Q let

- $adom(\mathbf{I})$:= the set of constants occurring in $\mathbf{I}$, the *active domain* of $\mathbf{I}$;
- $adom(Q)$:= the set of constants occurring in Q, the *active domain* of Q;
- $adom(Q, \mathbf{I}) := adom(Q) \cup adom(\mathbf{I})$, the *active domain* of Q and $\mathbf{I}$.

A set $\mathbf{d} \subseteq \mathbf{dom}$ is *admissible* for Q and $\mathbf{I}$ if $adom(Q, \mathbf{I}) \subseteq \mathbf{d}$.

Given an admissible $\mathbf{d}$ we define $\mathbf{I_d}$ similarly as $\mathbf{I}$, with the exception that

$$\Delta^{\mathbf{I_d}} = \mathbf{d}.$$

unibz

Query Semantics

Let $\mathbf{d}$ be admissible for $Q = \{\bar{x} \mid \phi\}$ and $\mathbf{I}$.

Then we define the *answer* of Q over $\mathbf{I}$ *relative to* $\mathbf{d}$ as

$$Q_{\mathbf{d}}(\mathbf{I}) = \{\alpha(\bar{x}) \mid \mathbf{I}_{\mathbf{d}}, \alpha \models \phi\}.$$

Intuitively, different semantics have different quantifier ranges.

The extreme cases are:

- *Natural semantics* $Q_{\mathsf{nat}}(\mathbf{I})$: unrestricted interpretation, that is $\mathbf{d} = \mathbf{dom}$
- *Active domain semantics* $Q_{\mathsf{adom}}(\mathbf{I})$: the range of quantifiers is the set of all constants in Q and in $\mathbf{I}$, that is $\mathbf{d} = \mathit{adom}(Q, \mathbf{I})$.

unibz

Domain Independence

Definition

Let Q be a RelCalc query. We say that Q is **domain independent** if

$$Q_{\mathbf{d}}(\mathbf{I}) = Q_{\mathsf{nat}}(\mathbf{I})$$

for all instances $\mathbf{I}$ and all domains $\mathbf{d} \subseteq \mathbf{dom}$ that are admissible for Q and $\mathbf{I}$.

Is domain independence decidable or not?
How can one prove this result?

What is the relationship between domain independent queries and relational algebra queries?

unibz

Equivalence Theorem of Relational Query Languages

The *domain-independent relational calculus* (DI-RelCalc) consists of all domain-independent calculus queries

Theorem (Codd)

Relational Algebra and DI-RelCalc have the same expressivity

That is, for every relational algebra expression E, there is a DI-RelCalc query Q such that $E \equiv Q$ and vice versa.

Remark: The theorem only holds for relational algebra with the operator

- singleton $\{d\}$,

where $d \in$ **dom**. Singleton models "$x = d$" in queries, as in

$$\{(x, y) \mid x = d \wedge P(y)\}.$$

unibz

Codd's Theorem: Proof Idea

We have shown that every algebra query can be translated into RelCalc. The converse statement follows from the next lemma.

Lemma

One can rewrite every RelCalc query Q_ϕ to a Relational Algebra query E_ψ such that for every instance $\mathbf{I}$ we have

$$Q_{\phi,adom(\phi,\mathbf{I})}(\mathbf{I}) = E_\phi(\mathbf{I}).$$

Proof Sketch.

Consider $Q_\phi = \{\bar{x} \mid \phi\}$. It is straightforward to define a unary algebra expression E_{adom} such that for all instances $\mathbf{I}$ we have

$$E_{adom}(\mathbf{I}) = adom(\phi, \mathbf{I}).$$

Then we translate every subformula ψ of ϕ into an expression E_ψ such that

$$E_\psi(\mathbf{I}) = (Q_\psi)_{adom(\phi,\mathbf{I})}.$$

Codd's Theorem: Proof Idea (cntd)

Proof Sketch (cntd).

We illustrate the translation with examples:

If $\psi(y_1, y_2) = R(d_1, y_1, y_2, d_2)$, where R has the schema $R(A_1, A_2, A_3, A_4)$,
then $E_\psi = \pi_{A_2,A_3}(\sigma_{A_1=d_1 \wedge A_4=d_2}(R))$.

If $\psi(y_1, y_2) = (y_1 \neq y_2)$, where A_1, A_2 correspond to y_1, y_2,
then $E_\psi = \sigma_{A_1 \neq A_2}(E_{adom} \times E_{adom})$.

If $\psi(y_1, y_2, y_3) = \psi'(y_1, y_2) \vee \psi'(y_2, y_3)$, where A_1, A_2, A_3 correspond to y_1, y_2, y_3,
then $E_\psi = (E_{\psi'} \times E_{adom}) \cup (E_{adom} \times E_{\psi''})$.

If $\psi(y_1, \ldots, y_m) = \neg\psi'(y_1, \ldots, y_m)$,
then $E_\psi = (E_{adom} \times \cdots \times E_{adom}) \setminus E_{\psi'}$.

If $\psi(y_2, \ldots y_m) = \exists y_1\, \psi'(y_1, \ldots, y_m)$, where $A_1, \ldots, A_m$ correspond to $y_1, \ldots, y_m$,
then $E_\psi = \pi_{A_2,\ldots,A_m}(E_{\psi'})$.

We also need renaming, e.g., to give a name to the attribute of E_{adom}. □

Safe-Range Queries

Safe range queries are a syntactically defined fragment of the relational calculus that contains *only* domain-independent queries

(and thus are also a fragment of DI-RelCalc)

One can show: Safe-Range RelCalc $\equiv$ DI-RelCalc

Steps in defining safe-range queries:

- a syntactic *normal form* of the queries
- a mechanism for determining whether a variable is *range restricted*

Then a query is safe-range iff all its free variables are range-restricted.

unibz

Safe-Range Normal Form (SRNF)

Equivalently rewrite query formula ϕ

- **Rename variables apart:** Rename variables such that each variable x is quantified at most once and has only free or only bound occurrences.
- **Eliminate $\forall$:** Rewrite $\forall x\, \phi \;\mapsto\; \neg\exists x\, \neg\phi$
- **Eliminate implications:** Rewrite $\phi \rightarrow \psi \;\;\mapsto\;\; \neg\phi \vee \psi$
  (and similarly for $\leftrightarrow$)
- **Push negation down as far as possible:** Use the rules
  - $\neg\neg\phi \;\mapsto\; \phi$
  - $\neg(\phi_1 \wedge \phi_2) \;\mapsto\; \neg\phi_1 \vee \neg\phi_2)$
  - $\neg(\phi_1 \vee \phi_2) \;\mapsto\; \neg\phi_1 \wedge \neg\phi_2)$
- *Flatten '$\exists$'s:* No child of an '$\exists$' in the formula parse tree is an '$\exists$'
  (this step is not essential)

unibz

Safe-Range Normal Form (2)

- The result of rewriting a query Q is called $\mathit{SRNF}(Q)$
- A query Q is in *safe-range normal form* if $Q = \mathit{SRNF}(Q)$

Examples:

$Q_1(\text{th}) = \exists\ \text{tl}\ \exists\ \text{dir}\ (\text{Movie(tl, dir,'Depp')} \wedge \text{Schedule(th,tl)})$

$\mathit{SRNF}(Q_1) = \exists\ \text{tl, dir}\ (\text{Movie(tl, dir,'Depp')} \wedge \text{Schedule(th,tl)})$

$Q_2(\text{dir}) = \forall\ \text{th}\ \forall\ \text{tl'}\ (\text{Schedule(th,tl')} \rightarrow \exists\ \text{tl}\ \exists\ \text{act}\ (\text{Schedule(th,tl)} \wedge \text{Movie(tl, dir, act)}))$

$\mathit{SRNF}(Q_2) = \neg\exists\ \text{th, tl'}\ (\text{Schedule(th,tl')} \wedge \neg\exists\ \text{tl, act}\ (\text{Schedule(th,tl)} \wedge \text{Movie(tl, dir, act)}))$

unibz

Range Restriction

Three elements:

- Syntactic condition on formulas in SRNF
- Intuition: all possible values of a variable lie in the active domain
- If a variable does not fulfill this, then the query is rejected

unibz

Algorithm Range Restriction (rr)

Input: formula ϕ in SRNF

Output: subset of the free variables of ϕ or $\bot$
(indicating that a quantified variable is not range restricted)

case ϕ **of**

$R(t_1, \ldots, t_n)$:	$rr(\phi) :=$ the set of variables from $t_1, \ldots, t_n$
$x = a$, $a = x$:	$rr(\phi) := \{x\}$
$\phi_1 \wedge \phi_2$:	$rr(\phi) := rr(\phi_1) \cup rr(\phi_2)$
$\phi_1 \wedge x = y$:	**if** $\{x, y\} \cap rr(\phi_1) = \emptyset$ **then** $rr(\phi) := rr(\phi_1)$ **else** $rr(\phi) := rr(\phi_1) \cup \{x, y\}$
$\phi_1 \vee \phi_2$:	$rr(\phi) := rr(\phi_1) \cap rr(\phi_2)$
$\neg\phi_1$:	$rr(\phi) := \emptyset$
$\exists x_1, \ldots, x_n \phi_1$:	**if** $\{x_1, \ldots, x_n\} \subseteq rr(\phi_1)$ **then** $rr(\phi) := rr(\phi_1) \setminus \{x_1, \ldots, x_n\}$ **else return** $\bot$

end case

Here, $S \cup \bot = \bot \cup S = \bot$ and similarly for $\cap, \setminus$

unibz

Algorithm Range Restriction (rr)/2

To better understand the rationale behind the definition of the set $rr(\phi)$, consider the following three formulas:

$$\psi_1(x) = (P(x) \wedge \neg S(y)) \vee R(x, y)$$
$$\psi_2(x) = P(x) \vee R(x, y)$$
$$\psi_3(x) = (P(x) \wedge \neg S(y)) \wedge R(x, y)$$

We have

$$rr(\psi_1) = \{x\},\ rr(\psi_2) = \{x\},\ rr(\psi_3) = \{x, y\}.$$

Considering the three formulas $\phi_i = \exists y\,(\psi_i)$, where $i = 1, 2, 3$, we see that ϕ_1 and ϕ_2 are not range restricted, while ϕ_3 is.

However, while ϕ_1 is domain-dependent, ϕ_2 is not.

Hence, the range-restricted formulas are a proper subset of the domain-independent ones.

unibz

Algorithm Range Restriction (rr)/2

To understand the rationale behind the definition of the set $rr(\phi)$, consider the following three queries:

$$\phi_1(x) = \exists y\,((P(x)) \vee R(x,y))$$
$$\phi_2(x) = \exists y\,((P(x) \wedge \neg S(y)) \vee R(x,y))$$
$$\phi_3(x) = \exists y\,((P(x) \wedge \neg S(y)) \wedge R(x,y))$$

We have

$$rr(\phi_1) = \{\}$$

unibz

Safe Range Theorem

Definition

Let ϕ be a formula and $\tilde{\phi}$ be its safe range normal form.

- Then ϕ is a *safe range formula* if the range restriction algorithm rr returns the set of free variables of $\tilde{\phi}$.
- A RelCalc query is a *safe range query* if it is defined by a safe range formula.

Theorem (Safe Range Theorem)

- All safe range queries are domain independent.
- For every domain independent query there is an equivalent safe range query.

The proof is very technical. We refer for it to the book [AHV95].

unibz

What Has This To Do With SQL?

We define the set of **nice SQL** queries as consisting of the queries constructed

- with `SELECT`, `FROM` and `WHERE` clauses
  plus `UNION` of subqueries
  plus nesting with `EXISTS` and `IN`
- with a `DISTINCT` in the `SELECT` clause
- where the `SELECT` clause contains only attributes
- with atomic conditions in `WHERE` clauses being equalities and comparisons, involving only constants and attributes
- with conditions in `WHERE` clauses being boolean combinations of atomic, `EXISTS`, and `IN` conditions

We call the set of all those queries *Nice SQL* (short **NSQL**)

unibz

Nice SQL and Relational Query Languages

Theorem

Relational algebra, DI-RelCalc, and NSQL have the same expressivity

This should not be surprising because

- NSQL combines the query constructs that have a correspondence in FOL
- We dropped, among others,
  - arithmetic ("+", "−", "∗"),
  - string functions, string matching,
  - null values, outer joins,
  - aggregation

unibz

Nice SQL: Exercise

Express the following queries over our university schema in NSQL

- Which are the names of students that have passed an exam in CS?
- What are the names of the courses for which student Egger has failed an exam?
- Which students have failed an exam for the same course at least twice?
- Which students (given by their id) have never failed an exam in CS?
- Which students (given by their id) have passed the exams for all courses in CS?

unibz

Looking Back ...

We have reviewed three formalisms for expressing queries

- Relational Algebra
- Relational Calculus (with its domain-independent fragment)
- Nice SQL

and seen that they have the same expressivity

However, crucial properties ((un)satisfiability, equivalence, containment) are undecidable

Hence, automatic analysis of such queries is impossible

Can we do some analysis if queries are simpler?

unibz