Information Integration

Part 1: Basics of Relational Database Theory

Werner Nutt

Faculty of Computer Science
Master of Science in Computer Science

A.Y. 2010/2011

Freie Universität Bozen
Libera Università di Bolzano
Free University of Bozen · Bolzano

Traditional Approach to Databases

- A single large repository of data
- Database administrator in charge of access to data
- Users interact with the database through application programs
- Programmers write those (embedded SQL, other ways of combining general purpose programming languages and DBMSs)
- Queries dominate; updates less common
- DMBS takes care of lots of things for you such as query processing and optimisation concurrency control enforcing database integrity

Motivational slides due to Leonid Libkin

Traditional Approach to Databases (contd)

- This model works very within a single organisation that either
  - does not interact much with the outside world, or
  - the interaction is heavily controlled by the DB administrators
- What do we expect from such a system?
  1. Data is relatively clean; little incompleteness
  2. Data is consistent (enforced by the DMBS)
  3. Data is there (resides on the disk)
  4. Well-defined semantics of query answering
     (if you ask a query, you know what you want to get)
  5. Access to data is controlled

The World is Changing

- The traditional model still dominates, but the world is changing
- Many huge repositories are publicly available
  - In fact, many are well-organised databases (e.g., imdb.com, the CIA World Factbook, many genome databases, the DBLP server of CS publications, etc etc etc)
- Many queries cannot be answered using a single source
- Often data from various sources needs to be combined, e.g.
  - company mergers
  - restructuring databases within a single organisation
  - combining data from several private and public sources

Theme of This Course

- Databases are everywhere these days
- Every enterprise has a database; they merge, combine data hence data integration
- In addition, a lot of data is available on the web, but often one needs many sources to answer a query
- Hence (almost) everyone needs to integrate data
- Huge investment from leading companies, IBM, Oracle, Microsoft
- Very ad hoc solutions; but finally we understand what the real problems in data integration are, and have some solutions (but not all!)

Topics

- Basics of Relational Database Theory
- Modeling Information Sources: Global as View, Local as View
- Query Semantics and Query Planning
- Sources with Access Limitations (Forms, Web Services)
- Data Exchange
- Schema Mapping
- Data Quality: Consistency and Completeness

Preliminary Course Overview

- No textbook, since none exists (but survey and research papers)
- Slides, papers, and links to further info will be posted on course website (reachable from my home page)
- Coursework: exercises and possibly implementation project
- Final mark $= \max \{$exam mark,
  $0.7 \times$ exam mark $+ 0.3 \times$ coursework mark$\}$
- Office hours: Tuesday, 2pm-4pm

Relational Databases: Principles

A database has two parts: schema and instance

The schema describes how data is organized:

- relations with their names, arity, names and types of attributes
- integrity constraints like key and foreign key constraints, functional dependencies, inclusion dependencies, check constraints

The instance contains the actual data:

- for every relation, there is a relation instance
- the relation instance is a set (multiset?) of tuples of the right arity and type

Often, we ignore types and integrity constraints
Sometimes, we ignore also the attribute names

Example Schema: Students and Courses

Relation schemas

`Student(sid: INTEGER, sname: STRING, city: STRING, age: INTEGER)`

`Course(cid: INTEGER, cname: STRING, faculty: STRING)`

`Enrolled(sid: INTEGER, cid: INTEGER, aY: STRING, mark: STRING)`

Integrity constraints

- Primary keys

  `Student(sid)`
  `Course(cid)`
  `Enrolled(sid, cid, aY)`

- Foreign keys:

  `Enrolled(sid)` references `Student(sid)`
  `Enrolled(cid)` references `Course(cid)`

Schemas: Formalization

A **relation schema** consists of

- a relation name
- an ordered list of attributes, possibly with types

Abstract notation $R(A_1, \ldots, A_n)$, or $R(A_1\colon \tau_1, \ldots, A_n\colon \tau_n)$

The *arity* of R, written *ary*(R), is the number of arguments of R

A **database schema** $\mathcal{S}$ consists of

- a *signature* Σ, which is a set of relation schemas
- a set Γ of *integrity constraints* over Σ,
  which may be expressed as formulas in first-order logic (FOL)

Simplified notation: $\mathcal{S} = \{R_1, \ldots, R_m\}$, or $\mathcal{S} = \{R_1/n_1, \ldots R_m/n_m\}$,
(i.e., we only mention the names or the names with their arity)

Exercise: Express the primary and foreign key constraints
in the Students and Course schema by FOL formulas

Domain: Formalization

We assume there is an infinite set of constants **dom**, called the **domain**

When we ignore types, we do not make any assumptions about the constants in **dom**

Otherwise, **dom** $= \bigcup_{i=1}^{k} \tau_i$, where $\tau_1, \ldots, \tau_k$ are the types

Definition

The order of a type τ is

- *dense* if for every $a, b \in \tau$ with $a < b$, there is a $c \in \tau$ such that $a < c < b$
- *discrete* if for every a, $b \in \tau$ with $a < b$, there are at most finitely many c such that $a < c < b$

Example

Consider integers, reals, strings, and booleans.
Which type has a dense and which a discrete ordering?

Relation Instances

Relation R with arity n:

- an instance of R is a finite set of n-tuples over **dom**

Relation R with schema $R(A_1\colon \tau_1, \ldots, A_n\colon \tau_n)$:

- as before, plus the components of the n-tuples in an instance have to be of the right type

Schema Instances

An **instance of the signature** Σ is a function $\mathbf{I}$ that

- maps every $R \in \Sigma$ to an instance of R, denoted $\mathbf{I}(R)$

Every instance $\mathbf{I}$ of Σ can be seen as a **first-order interpretation/structure** (also denoted $\mathbf{I}$):

- domain of $\mathbf{I}$ is $\Delta^{\mathbf{I}} = \mathbf{dom}$
- $c^{\mathbf{I}} = c$, for every $c \in \mathbf{dom}$
  (proper names, i.e., every constant is interpreted as itself)
- $R^{\mathbf{I}} = \mathbf{I}(R)$

A function $\mathbf{I}$ is an **instance of the schema** $\mathcal{S} = (\Sigma, \Gamma)$ if

- $\mathbf{I}$ is an instance of Σ
- $\mathbf{I}$ satisfies every integrity constraint $\gamma \in \Gamma$ in the sense of first-order logic (FOL)

Logic Programming Perspective

Often an alternate definition of instances is helpful

Definition

- A *fact* over a relation R with arity n is an expression $R(a_1, \ldots, a_n)$, where $a_1, \ldots, a_n \in$ **dom**
- A *relation instance* is a finite set of facts over R
- A *schema instance* $\mathbf{I}$ of Σ is a finite set of facts over the relations in Σ

Example

$\mathbf{I}_{\text{univ}} = \{$ `Student`(123, Egger, Bozen, 24), `Student`(777, Hussein, Dresden, 22),
`Course`(104, Programming, CS), `Course`(106, Databases, CS),
`Course`(217, Optics, PHYS)
`Enrolled`(123, 104, 07/08, pass), `Enrolled`(123, 106, 09/10, fail),
`Enrolled`(123, 106, 08/01, fail), `Enrolled`(123, 106, 10/11, pass),
`Enrolled`(777, 217, 09/10, pass) $\}$

Relational Queries

A **query** over a schema $\mathcal{S}$ is

- a **function** that maps every instances of $\mathcal{S}$ to a set of tuples such that
  - all tuples have the same length (= arity of the query)
  - tuple values at the same position have the same type
- a **piece of syntax** that defines such a function

Query languages are/should be **declarative**:

- you express what you want to know, not how to compute it
  (a query engine analyzes the query and creates an execution plan)

Relational Query Languages

- Theoretical languages
  - Relational Algebra (that's how Codd started it)
  - Relational Calculus (= FOL in essence)
  - Datalog (drops negation, adds recursion)
- Commercial language: SQL
  - = Relational Calculus (at its core)
  - \+ Relational Algebra
  - \+ a bit of Datalog (implemented in IBM DB2, Microsoft SQL Server)
  - \+ aggregates, arithmetic, nulls, . . . , functions, procedures

Relational Algebra

Expressions E are built up from

- relation symbols R

using the operators

- union $(E_1 \cup E_2)$, intersection $(E_1 \cap E_2)$, set difference $(E_1 \setminus E_2)$, called boolean operators
- selection $\sigma_C(E)$
- projection $\pi_X(E)$
- cartesian product $E_1 \times E_2$
- join $E_1 \bowtie_C E_2$
- attribute renaming $(\rho_{A \leftarrow B}(E))$

where C is a condition involving equalities and comparisons between attributes and constants, and X is a set of attributes of E

For an instance $\mathbf{I}$, an expression E is evaluated as a set of tuples $E(\mathbf{I})$

A **query** is an **expression**

Exercises

Express the following queries over our university schema in Relational Algebra

- 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 have never failed an exam in Physics?

Evaluate the expressions over the instance $\mathbf{I}_{\text{univ}}$

Relational Calculus Queries

Definition

A **query** in (domain) relational calculus (RelCalc) has the form

$$Q = \{(x_1, \ldots, x_n) \mid \phi\}$$

where

- ϕ is a predicate logic formula
- $x_1, \ldots, x_n$ are the free variables of ϕ

We say that

- ϕ is the **body** of the query,
- $x_1, \ldots, x_n$ are the **output variables**, and
- n is the **arity** of the query.

If the arity is not important, we write $\bar{x}$ instead of $x_1, \ldots, x_n$

We sometimes write Q_ϕ to denote the query defined by ϕ

Reminder on Predicate Logic Formulas

A *term* is a constant or a variable

An *atom* is an expression $R(t_1, \ldots, t_n)$ where R is a relation symbol of arity n and $t_1, \ldots, t_n$ are terms

A *formula* F is an atom or has the form

- $F_1 \wedge F_2$, $F_1 \vee F_2$, or $F_1 \rightarrow F_2$
- $\neg F$
- $\exists\, x\, F(x)$, $\forall\, x\, F(x)$

where F, F_1, F_2 are formulas

Exercise: Show that the logical symbols $\wedge$, $\exists$, $\neg$ suffice to express all other symbols

Equality and Built-in Predicates

Sometimes we use also the predicate symbols

$$\text{“}{=}\text{”}, \quad \text{“}{<}\text{”}, \quad \text{“}{\leq}\text{”}, \quad \text{“}{\neq}\text{”}$$

Atoms with these symbols are called

- equalities (“$=$”)
- comparisons (“$<$”, “$\leq$”)
- disequalities (“$\neq$”)

Clearly, they can only be applied to terms of the same type

Comparisons can only be used for terms of a type that is linearly ordered

Bound and Free Variables

Definition

- An occurrence of a variable x in formula ϕ is *bound*
  if it is within the scope of a quantifier $\exists x$ or $\forall x$
- An occurrence of a variable in ϕ is *free* iff it is not bound
- A variable of formula ϕ is *free* if it has a free occurrence

Free variables specify the output of a query

Relational Calculus Queries: Semantics

In FOL, the semantics of a formula is defined in terms of *interpretations* and *assignments*. Recall:

- every instance $\mathbf{I}$ defines a first-order interpretation $\mathbf{I}$
- an assignment is a mapping $\alpha\colon$ **var** $\rightarrow$ **dom**

There is a classical recursive definition of when an interpretation $\mathbf{I}$ and an assignment α *satisfy* a formula ϕ, written

$$\mathbf{I}, \alpha \models \phi,$$

which we take for granted

Definition

Let $Q = \{(x_1, \ldots, x_n) \mid \phi\}$ be a query. We define the **answer** of Q over $\mathbf{I}$ as

$$Q(\mathbf{I}) = \{\alpha(\bar{x}) \mid \mathcal{I}, \alpha \models \phi\}$$

Exercise

Express the following queries over our university schema in Relational Calculus

- Which are the names of students that have passed an exam in CS?
- 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?

Evaluate the expressions over the instance $\mathbf{I}_{\text{univ}}$

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. Exercise! :-) □

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

What about the converse statement?

Safe Queries

Proposition

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

Proof.

How? $\square$

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?

Negation and Safety

Consider

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

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

Theorem

Safety of relational calculus queries is undecidable

Proof.

Idea: Encode the finite satisfiability problem for FOL, which is known to be undecidable (Trakhtenbrot's Theorem) □

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 $Q_1(\mathbf{I}) = Q_2(\mathbf{I})$ for all instances $\mathbf{I}$

Theorem

Satisfiability, equivalence, and containment are undecidable for RelCalc queries

Proof.

Undecidability of satisfiability follows immediately from Trakhtenbrot's theorem about undecidability of finite satisfiability (although it is not exactly the same). The other two claims can then be shown by reduction. Exercise! □

Domain Independence

Consider the query

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

Q is safe (only `Persons` are returned)

However, for arbitrary interpretations, the answer to Q depends on the domain over which $\forall\, y$ ranges

A query with this property is **domain dependent**, otherwise **domain independent**

You guess whether domain independence is decidable or not, and how one can prove this result :-)

Equivalence Theorem of Relational Query Languages

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

Theorem

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.

One can define the decidable class of **safe range** *queries, which has the property that for every domain-independent query there is an equivalent safe-range query*

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**)

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

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?

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?

Many Natural Queries Can Be Expressed ...

...in SQL

- using only a *single* `SELECT-FROM-WHERE` *block* and *conjunctions* of atomic conditions in the `WHERE` clause;
- we call these the **CSQL queries**.

...in Relational Algebra

- using only the operators selection $\sigma_C(E)$, projection $\pi_C(E)$, join $E_1 \bowtie_C E_2$, renaming $(\rho_{A \leftarrow B}(E))$;
- we call these the **SPJR queries** (= select-project-join-renaming queries)

...in Relational Calculus

- using only the logical symbols "$\wedge$" and $\exists$ such that every variable occurs in a relational atom;
- we call these the **conjunctive queries**

Conjunctive Queries

Theorem

The classes of CSQL queries, SPJR queries, and conjunctive queries have all the same expressivity. Queries can be equivalently translated from one formalism to the other in polynomial time.

Proof.

By specifying translations. □

Intuition: By a conjunctive query we define a pattern of what the things we are interested in look like. Evaluating a conjunctive query is matching the pattern against the database instance.

Rule Notation for Conjunctive Queries

By pulling the quantifiers outside, every conjunctive calculus query can be written as

$$Q = \{(x_1, \dots, x_k) \mid \exists\, y_1, \dots, \exists\, y_l\, (A_1 \land \dots \land A_m)\},$$

where $A_1, \dots, A_m$ are (relational and built-in) atoms

We say that $x_1, \dots, x_k$ are the **distinguished variables** of Q
and $y_1, \dots, y_m$ the **nondistinguished variables**

We will often write such a query, using a rule in the style of PROLOG, as

$$Q(\bar{x}) \coloneq R_1(\bar{}), \dots, A_m$$

We say $Q(x_1, \dots, x_k)$ is the **head** of the query and $A_1 \land \dots \land A_m$ the **body**

Note: Existential quantifiers are implicit,
since we list the free variables in the head.

Semantics of Conjunctive Queries

Consider a **conjunctive formula**

$$\phi = \exists y_1, \dots, y_l (A_1 \wedge \cdots \wedge A_m)$$

such that

- $A_1, \dots, A_m$ are atoms, with relational or built-in predicates
- $\bar{x} = (x_1, \dots, x_k)$ is the tuple of free variables of ϕ
- every variable occurs in a relational atom

Then Q_ϕ is a conjunctive query

Proposition (Answer Tuple for a Calculus Query)

Let $\mathbf{I}$ be an instance. A k-tuple of constants $\bar{c}$ is an **answer tuple** for Q_ϕ over $\mathbf{I}$ if and only if there is an assignment α such that

- $\bar{c} = \alpha(\bar{x})$
- $\mathbf{I}, \alpha \models A_j$ for $j = 1, \dots, m$

Schematic Notation of Conjunctive Queries

$$Q(\bar{x}) \mathrel{:\!-} L,\, M,$$

where

- $L = R_1(\bar{t}_1), \ldots, R_n(\bar{t}_n)$ is a conjunction of relational atoms
- $M = B_1, \ldots, B_p$ is a conjunction of built-in atoms (that is, with predicates "$<$", "$\leq$", "$\neq$"),
- every variable occurs in some $R_j(\bar{t}_j)$ *(guarantees safety of Q!)*

Proposition (Answer Tuple for a Rule)

The tuple $\bar{c}$ is an answer for Q over $\mathbf{I}$ iff there is an assignment α such that

- $\bar{c} = \alpha(\bar{x})$
- $\alpha(\bar{t}_j) \in \mathbf{I}(R_j)$, for $j = 1, \ldots, n$
- $\alpha \models M$

Conjunctive Queries: Logic Programming (LP) Perspective

$\mathbf{I}$ finite set of ground facts (= instance in LP perspective)

Proposition (Answer Tuple in LP Perspective)

The tuple $\bar{c}$ is an answer for $Q(\bar{x}) \mathrel{:-} L,\, M$ over $\mathbf{I}$
iff there is an assignment α for the variables of ϕ such that

- $\bar{c} = \alpha(\bar{x})$
- $\alpha(L) \subseteq \mathbf{I}$
- $\alpha \models M$

Note that for relational conjunctive queries (i.e., w/o built-ins), satisfaction of Q by α over $\mathbf{I}$ boils down to

$$\alpha(L) \subseteq \mathbf{I}$$

Elementary Properties of Conjunctive Queries

Proposition (Properties of Conjunctive Queries)

Let $Q(\bar{x}) \mathrel{:\!-} L, M$ be a conjunctive query. Then

1. the answer set $Q(\mathbf{I})$ is **finite** for all instances $\mathbf{I}$
2. Q is **monotonic**, that is,
$$\mathbf{I} \subseteq \mathbf{J} \text{ implies } Q(\mathbf{I}) \subseteq Q(\mathbf{J}) \text{ for all instances } \mathbf{I}, \mathbf{J}$$
3. Q is **satisfiable** if and only if M is satisfiable

Proof.

1. Holds due to safety condition and finiteness of $\mathbf{I}$
2. Follows easily with LP perspective
3. Exercise!

□

Evaluation of Conjunctive Queries: Decision Problems

How difficult is it to compute $Q(\mathbf{I})$?

Definition (Evaluation problem for a single conjunctive query Q)

Given: instance $\mathbf{I}$, tuple $\bar{c}$

Question: is $\bar{c} \in Q(\mathbf{I})$?

Definition (Evaluation problem for the class of conjunctive queries)

Given: conjunctive query Q, instance $\mathbf{I}$, tuple $\bar{c}$

Question: is $\bar{c} \in Q(\mathbf{I})$?

Note:

First problem: Q is fixed (**Data Complexity**)

Second problem: Q is part of the input (**Combined Complexity**)

Reminder on the Class NP

NP = the class of problems that can be decided by a nondeterminstic Turing machine in polynomial time.

We compare problems in terms of reductions:
For two problems $P_1 \in \Sigma_1^*$, $P_2 \in \Sigma_2^*$, a function $f\colon \Sigma_1^* \to \Sigma_2^*$ is a polynomial time **many-one reduction** (or Karp reduction) of P_1 to P_2 if and only if

- $s_1 \in P_1 \quad \Leftrightarrow \quad f(s_1) \in P_2$ for all $s_1 \in \Sigma_1^*$
- f can be computed in polynomial time

We write $P_1 \leq_m P_2$ if there is a Karp reduction from P_1 to P_2.
The relation "$\leq_m$" is a preorder (= reflexive, transitive relation)

Theorem (Cook, Karp)

There are problems in NP that are maximal wrt "$\leq_m$".

These problems are called **NP-complete.**

Evaluation of Conjunctive Queries: Complexity

Proposition (Data Complexity)

For every conjunctive query Q, there is a polynomial p, such that the evaluation problem can be solved in time $O(p(|\mathbf{I}|))$.

Idea: Q can be written as a selection applied to a cartesian product. What is the width of the cartesian product?

Hence, data complexity is in PTIME. Actually, data complexity of evaluating arbitrary FO (i.e., algebra or calculus) queries is in LOGSPACE

Proposition (Combined Complexity)

Given $Q(\bar{x})$:– L, M, $\mathbf{I}$ and $\bar{c}$, one can **guess** in linear time an α such that

- α satisfies L, M over $\mathbf{I}$
- $\alpha(\bar{x}) = \bar{c}$

Hence, combined complexity is in NP. Is it also NP-hard?

The 3-Colorability Problem

Definition (3-Colorability of Graphs)

Instance: A graph $G = (V, E)$

Question: Can G be colored with the three colours $\{r, g, b\}$ in such a way that two adjacent vertices have a distinct colour?

The 3-colorability problem is NP-complete

A graph G is 3-colourable if and only if there is a graph homomorphism from G to the simplex S_3, which consists of three vertices that are connected to each other

Reducing 3-Colorability to Evaluation

Theorem (Reduction)

There is a database instance $\mathbf{I}_{3col}$ such that for every finite graph G one can compute in linear time a relational conjunctive query $Q_G() \mathrel{:-} L$ such that

$$G \text{ is 3-colorable} \qquad \text{if and only if} \qquad Q_G(\mathbf{I}_{3col}) = \{()\}$$

Remark (Boolean Queries)

- A query without distinguished variables is called a *boolean* query
- Over an instance, a boolean query returns the empty tuple (), or nothing

This will shows NP-hardness of combined complexity of conjunctive query evaluation

The Reduction

Given graph $G = (V, E)$, where

$V = \{v_1, \ldots, v_n\}$ and

$E = \{(v_{i_l}, v_{j_l}) \mid i_l < j_l,\ 1 \leq l \leq m\}$

We construct $\mathbf{I}_{3col}$ and Q_G as follows

$$\mathbf{I}_{3col} = \{\mathsf{e}(r, b),\ \mathsf{e}(b, r),\ \mathsf{e}(r, g),\ \mathsf{e}(g, r),\ \mathsf{e}(b, g),\ \mathsf{e}(g, b)\}$$

$$Q_G() \coloneq \mathsf{e}(y_{i_1}, y_{j_1}), \ldots, \mathsf{e}(y_{i_m}, y_{j_m})$$

where $y_1, \ldots, y_n$ are new variables and
there is one atom $\mathsf{e}(y_{i_l}, y_{j_l})$ for each edge $(v_{i_l}, v_{j_l}) \in E$

Clearly, there is an $\alpha\colon \{y_1, \ldots, y_n\} \to \{r,\ g,\ b\}$ satisfying Q_G over $\mathbf{I}_{3col}$ iff there is a graph homomorphism from G to S_3

Evaluation of Conjunctive Queries in Practice

- To assess the practical difficulty of query evaluation, one usually looks only at **data complexity**: the size of the query is (very!) small compared to the size of the data
- **Query optimizers** try to find plans that minimize the cost of executing conjunctive queries:
  - Find a good **ordering of joins**
  - Identify the best **access paths** to data (indexes)

  The DBMS keeps **statistics** about size of relations and distribution of attribute values to estimate the cost of plans
- Well understood for a single DBMS,
  more **difficult if data sources are distributed**
  - often, info about access paths and statistics are missing in data integration scenarios
  - need to change execution plans on the fly

The 3-Satisfiability Problem

Ingredients

- Propositions $p_1, \ldots, p_n, \ldots$
- Literals l: proposition (p) or negated propositions ($\neg p$)
- 3-Clauses C: disjunctions of three literals $(l_1 \vee l_2 \vee l_3)$

Definition (3-Satisfiability)

Given: a finite set $\mathcal{C}$ of 3-clauses

Question: is $\mathcal{C}$ satisfiable, i.e., is there a truth assignment α such that α makes at least one literal true in every $C \in \mathcal{C}$?

The 3-Sat Problem is **the** classical NP-complete problem

Later on, we will use a reduction of 3-Satisfiability to Evaluation ...

Alternate Reduction From 3-Satisfiability

Theorem

For every set of 3-clauses $\mathcal{C}$, there is an instance $\mathbf{I}_\mathcal{C}$ and a boolean relational query $Q_\mathcal{C}$ such that

$$\mathcal{C} \text{ is satisfiable} \qquad \text{if and only if} \qquad Q_\mathcal{C}(\mathbf{I}_\mathcal{C}) \neq \emptyset$$

Definition of $\mathbf{I}_\mathcal{C}$ and $Q_\mathcal{C}$.

Let $\mathcal{C} = \{C_1, \ldots, C_m\}$ and consider propositions as variables.

- For every clause $C_i \in \mathcal{C}$, choose a relation symbol R_i.
- Let $p_1^{(i)}$, $p_2^{(i)}$, $p_3^{(i)}$ be the propositions in the clause C_i.
- Let $T_i = \{\bar{t}_1^{(i)}, \ldots, \bar{t}_7^{(i)}\}$ be the seven triples of truth values that satisfy C_i. E.g., if $C_i = p_2 \vee \neg p_4 \vee p_7$, then $T_i = \{0,1\}^3 \setminus \{(0,1,0)\}$.
- Define $\mathbf{I}_\mathcal{C} = \bigcup_i \{R_i(\bar{t}) \mid \bar{t} \in \mathcal{T}_i\}$.
- Define $Q_\mathcal{C}() \text{ :- } R_1(p_1^{(1)}, p_2^{(1)}, p_3^{(1)}), \ldots, R_m(p_1^{(m)}, p_2^{(m)}, p_3^{(m)})$.

.it

Properties of Conjunctive Queries

Satisfiability can be decided in PTIME, since

satisfiability of a conjunction of comparisons can be decided in PTIME

If we can decide containment, then we can also decide equivalence, since

$$Q_1 \equiv Q_2 \quad \text{if and only if} \quad Q_1 \sqsubseteq Q_2 \text{ and } Q_2 \sqsubseteq Q_1$$

If we can decide equivalence, we can also decide containment, since

$$Q_1 \sqsubseteq Q_2 \quad \text{if and only if} \quad Q_1 \equiv Q_1 \cap Q_2$$

Why is $Q_1 \cap Q_2$ again a conjunctive query?

We will concentrate on containment

Conjunctive Query Containment: Warm-Up

Find all containments and equivalences among the following conjunctive queries:

$$Q_1(x,y) \mathrel{:\!-} R(x,y),\ R(y,z),\ R(z,w)$$

$$Q_2(x,y) \mathrel{:\!-} R(x,y),\ R(y,z),\ R(z,u),\ R(u,w)$$

$$Q_3(x,y) \mathrel{:\!-} R(x,y),\ R(z,u),\ R(v,w),\ R(x,z),\ R(y,u),\ R(u,w)$$

$$Q_4(x,y) \mathrel{:\!-} R(x,y),\ R(y,3),\ R(3,z),\ R(z,w)$$

Idea: Reduce Containment to Evaluation! (1)

$$Q'(x,y) \coloneq R(x,y),\, R(y,z),\, R(y,u)$$

$$Q(x,y) \coloneq R(x,y),\, R(y,z),\, R(w,z)$$

Step 1 Turn Q into an instance $\mathbf{I}_Q$ by "freezing" the body of Q, i.e., replace variables x, y, z, w with constants c_x, c_y, c_z, c_w:

$$\mathbf{I}_Q = \{R(c_x, c_y),\, R(c_y, c_z),\, R(c_w, c_z)\}$$

Observe that $(c_x, c_y) \in Q(\mathbf{I}_Q)$

Idea: $\mathbf{I}_Q$ *is prototypical for any database where* Q *returns a result*

Step 2 Evaluate Q' over $\mathbf{I}_Q$

Case 1 If $(c_x, c_y) \notin Q'(\mathbf{I}_Q)$, then we have found a counterexample: $Q \not\sqsubseteq Q'$.

Idea: Reduce Containment to Evaluation! (2)

Case 2 If $(c_x, c_y) \in Q'(\mathbf{I}_Q)$, then there is an α such that

- $\alpha(x) = c_x, \quad \alpha(y) = c_y$
- $\alpha(A) \in \mathbf{I}_Q$ for every atom A' in the body of Q'

For instance,

$$\alpha = \{x/c_x,\, y/c_y,\, z/c_z,\, u/c_z\}$$

does the job.

With α we can extend every satisfying assignment for Q to a satisfying assignment for Q', as follows:

Let $\mathbf{I}$ be an arbitrary db instance and $(d, e) \in Q(\mathbf{I})$ be an answer of Q over $\mathbf{I}$. Then there is an assignment β such that

- $\beta(x) = d, \quad \beta(y) = e$
- $\beta(B) \in \mathbf{I}$ for every atom B in the body of Q.

Idea: Reduce Containment to Evaluation! (3)

Define the substitution α' (= mapping from terms to terms, not moving constants) by "melting" α, that is,
replacing every constant c_v with the corresponding variable v:

$$\alpha' = \{x/x,\, y/y,\, z/z,\, u/z\}.$$

Define $\beta' = \beta \circ \alpha'$, that is, as composition of first α' and then β.

Then $\beta'(x) = \beta(\alpha'(x)) = \beta(x) = d$ and, similarly, $\beta'(y) = e$.

Moreover if A' is an atom of Q', then

- $\alpha'(A')$ is an atom of Q, since $\alpha(A') \in \mathbf{I}_Q$
- $\beta'(A') = \beta(\alpha'(A)) \in \mathbf{I}$, since β maps every atom of Q to a fact in $\mathbf{I}$

Hence, $(d, e) = (\beta'(x), \beta'(y))$ is an answer of Q' over $\mathbf{I}$.

This shows, $Q(\mathbf{I}) \subseteq Q'(\mathbf{I})$ for an arbitrary $\mathbf{I}$ and thus, $Q \sqsubseteq Q'$.

Query Homomorphisms

Definition

Consider conjunctive queries without built-ins

$$Q'(\bar{x}) \mathrel{:-} L'$$

$$Q(\bar{x}) \mathrel{:-} L$$

A mapping $\delta\colon \mathit{Terms}(Q') \to \mathit{Terms}(Q)$ is a **query homomorphism** (from Q' to Q) if

- $\delta(c) = c$ for every constant c
- $\delta(x) = x$ for every distinguished variable x of Q'
- $\delta(L') \subseteq L$

Intuitively,

- δ respects constants and distinguished variables
- δ maps conditions of Q' to conditions in Q that are no less strict

Finding Homomorphisms

Find all homomorphisms among the following conjunctive queries:

$$Q_1(x, y) \coloneq R(x, y),\ R(y, z),\ R(z, w)$$

$$Q_2(x, y) \coloneq R(x, y),\ R(y, z),\ R(z, u),\ R(u, w)$$

$$Q_3(x, y) \coloneq R(x, y),\ R(z, u),\ R(v, w),\ R(x, z),\ R(y, u),\ R(u, w)$$

$$Q_4(x, y) \coloneq R(x, y),\ R(y, 3),\ R(3, z),\ R(z, w)$$

In terms of complexity, how difficult is it to decide whether there exists a homomorphism between two queries?

The Homomorphism Theorem

Theorem (Chandra/Merlin)

Let $Q'(\bar{x})$:– L' and $Q(\bar{x})$:– L be conjunctive queries (w/o built-in predicates). Then the following are equivalent:

- there exists a homomorphism from Q' to Q
- $Q \sqsubseteq Q'$.

Proof.

Straightforward by generalizing the previous example. □

What are homomorphisms for queries with built-in predicates?
What should we do with the comparisons?

Homomorphisms between Queries with Comparisons

Example

$$
\begin{aligned}
Q'() \mathrel{:\!-} & \ R(x, y), \\
& \quad x \leq 2,\ y \geq 3 \\
Q() \mathrel{:\!-} & \ R(u, v),\ R(v, w) \\
& \quad u \geq 3,\ v \geq 0,\ v \leq 1,\ w \geq 4
\end{aligned}
$$

There are two "relational" homomorphism s:

$$
\begin{aligned}
\delta = & \{x/u,\ y/v\} \\
\eta = & \{x/v,\ y/w\}
\end{aligned}
$$

Which of the two deserves the title of homomorphism?

Query Homomorphisms

Definition

Consider conjunctive queries with comparisons

$$Q'(\bar{x}) :\!-\ L', M'$$

$$Q(\bar{x}) :\!-\ L, M$$

A mapping $\delta\colon$ *Terms*$(Q') \to$ *Terms*(Q) is a **query homomorphism** if

- $\delta(c) = c$ for every constant c
- $\delta(x) = x$ for every distinguished variable x of Q'
- $\delta(L') \subseteq L$
- $M \models \delta(M')$.

Intuition: With respect to δ, the comparisons in Q are more restrictive than those in Q

Homomorphisms between Queries with Comparisons

Example

$$Q'(x) \coloneq P(x,y),\ R(y,z),$$
$$y \leq 3$$
$$Q(x) \coloneq P(x,w),\ P(x,x),\ R(x,u),$$
$$w \geq 5,\ x \leq 2$$

The substitution

$$\delta = \{x/x,\ y/x,\ z/u\}$$

- is a relational homomorphism
- satisfies $w \geq 5,\ x \leq 2 \models \delta(y) \leq 3$

Does the Hom Theorem Hold for Queries w/ Comparisons?

$$Q'(\bar{x}) \coloneq L', M' \qquad\qquad Q(\bar{x}) \coloneq L, M$$

Let $\delta\colon Q' \to Q$ be an hom, $\mathbf{I}$ an instance. Suppose $\bar{c} \in Q(\mathbf{I})$. Is $\bar{c} \in Q'(\mathbf{I})$?

Since $\bar{c} \in Q'(\mathbf{I})$, there is α such that

- $\alpha(\bar{x}) = \bar{c}$
- $\alpha(L) \subseteq \mathbf{I}$
- $\alpha \models M$.

Define $\alpha' = \alpha \circ \delta$. Then

- $\alpha'(\bar{x}) = \alpha(\delta(\bar{x})) = \alpha(\bar{x}) = \bar{c}$
- $\alpha'(L') = \alpha(\delta(L')) \subseteq \alpha(L) \subseteq \mathbf{I}$
- $\alpha \models \delta(M')$, since $\alpha \models M$ and $M \models \delta(M') \quad \Rightarrow \quad \alpha \circ \delta \models M'$.

Thus, $\bar{c} \in Q'(\mathbf{I})$.

The Homomorphism Theorem for Queries w/ Comparisons

We have just proved the following theorem:

Theorem (Homomorphisms Are Sufficient for Containment)

Let $Q'(\bar{x}) \mathrel{:\!-} L', M'$ and $Q(\bar{x}) \mathrel{:\!-} L, M$ be conjunctive queries.

If there is a homomorphism from Q' to Q, then $Q \sqsubseteq Q'$.

Does the Converse Hold as Well?

Intuitition:

- Blocks can be either black or white.
- Block 1 is on top of block 2, which is on top of block 3.
- Block 1 is white and block 3 is black.
- Is there a white block on top of a black block?

Example

$$Q'() \text{ :- } S(x, y), x \leq 0, y > 0$$

$$Q() \text{ :- } S(0, z), S(z, 1)$$

Case Analysis for Q

Define

$$Q_{\{z<0\}}() \coloneq S(0,z), S(z,1), z<0$$

$$Q_{\{z=0\}}() \coloneq S(0,0), S(0,1)$$

$$Q_{\{0<z<1\}}() \coloneq S(0,z), S(z,1), 0<z, z<1$$

$$Q_{\{z=0\}}() \coloneq S(0,1), S(1,1)$$

$$Q_{\{1<z\}}() \coloneq S(0,z), S(z,1), z>1$$

We note

- Q is equivalent to the union of $Q_{\{z<0\}}, \ldots, Q_{\{1<z\}}$
- there is a homomorphism from Q' to $Q_{\{\ldots\}}$ for each ordering $\{\ldots\}$
- $Q_{\{\ldots\}} \sqsubseteq Q'$ for for each $\{\ldots\}$

$\Rightarrow$ $Q \sqsubseteq Q'$

Idea: Replace Q with $\bigcup_{\{\ldots\}} Q_{\{\ldots\}}$ when checking "$Q \sqsubseteq Q'$?"

Linearizations

We now make this idea formal.

- We assume that all of **dom** is one linearly ordered type. Let
  - D be a set of constants from **dom**,
  - W be a set of variables,
  - and let $T := D \cup W$ denote their union.

- A **linearization** of T over **dom** is a set of comparisons N over the terms in T such that for any s, $t \in T$ exactly one of the following holds:
  - $N \models_{\mathbf{dom}} s < t$
  - $N \models_{\mathbf{dom}} s = t$
  - $N \models_{\mathbf{dom}} s > t$.

- That is, N partitions the terms into classes such that
  - the terms in each class are equal and
  - the classes are arranged in a strict linear order

Linearizations (cntd)

- **Remark:** A class of the induced partition contains at most one constant
- **Remark:** Whether or not N is a linearization may depend on the domain. Consider e.g.,

$$\{1 < x,\ x < 2\}$$

- A linearization N of T over **dom** is **compatible** with a set of comparisons M if $M \cup N$ is satisfiable over **dom**

Linearizations of Conjunctive Queries

- When checking containment of two queries, we have to consider linearizations that contains the constants of *both* queries
- Let
$$Q(\bar{x}) \coloneq L,\, M$$
be a query and
  - W be the set of variables occurring in Q
  - D be a set of constants that comprise the constants of Q
- Then we denote with $\mathcal{L}_D(Q)$ the set of all linearizations of $D \cup W$ that are compatible with the comparisons M of Q

Linearizations of Conjunctive Queries (cntd)

Proposition

Let Q, W, D and M be as above and let $\alpha\colon W \to$ **dom** be an assignment. Then the following are equivalent:

- $\alpha \models M$
- $\alpha \models N$ for some $N \in \mathcal{L}_D(Q)$

Proof.

"$\Leftarrow$" Let $N \in \mathcal{L}_D(Q)$. Since *Terms*$(M) \subseteq D \cup W$, and N is a linearization of $D \cup W$, we have that $N \models M$:
To see this, let $s \leq t \in M$. Then $N \models s < t$ or $N \models s = t$ or $N \models s > t$.
Since $M \cup \{s > t\}$ is unsatisfiable, we have $N \models s \leq t$.

"$\Rightarrow$" For $\alpha \models M$ let $N_\alpha = \{B \mid B$ is a built-in atom with terms from $D \cup W$ and $\alpha \models B\}$.
Then N_α is a linearization of $D \cup W$ compatible with M and $\alpha \models N_\alpha$. □

Linearizations of Conjunctive Queries (cntd)

Let Q be as above. Let N be a linearization of $T = D \cup W$ compatible with M.

- Note: N defines an equivalence relation on T, where each equivalence class contains at most one constant
- A substitution ϕ is *canonical* for N if
  - it maps all elements in an equivalence class of N to one term of that class
  - if a class contains a constant, then it maps the class to that constant.
- Then Q_N is obtained from Q by means of a canonical substitution ϕ for N as

$$Q_N(\phi(\bar{x})) \coloneq \phi L \wedge \phi N,$$

  that is,
  - we first replace M with N
  - and then "eliminate" all equalities by applying ϕ
- **Note:** We must admit also queries with a tuple of terms $\bar{s}$ in the head

Linearizations of Conjunctive Queries (cntd)

Definition (Linearization)

The queries

$$Q_N(\phi(\bar{x})) \coloneq \phi L \land \phi N,$$

are called **linearizations** of Q w.r.t. N

- There may be more than one linearization of Q w.r.t. N, but all linearizations are identical up to renaming of variables
- Note that ϕ is a homomorphism from Q to Q_N

Linear Expansions

Definition (Linear Expansion)

A **linear expansion** of Q over D is a family of queries $(Q_N)_{N \in \mathcal{L}_D(Q)}$, where each Q_N is a linearization of Q w.r.t. N

If Q and D are clear from the context we write simply $(Q_N)_N$.

Proposition

Let $(Q_N)_N$ be a linear expansion of Q over D.
Then Q and the union $\bigcup_{N \in \mathcal{L}_D(Q)} Q_N$ are equivalent.

Proof.

Follows from two facts:

- M and the disjunction $\bigvee_{N \in \mathcal{L}_D(Q)} N$ are equivalent
- If ϕ is a canonical substitution for N, then $Q_N(\phi(\bar{x})) \coloneq \phi L, \phi N$ and $Q(\bar{x}) \coloneq L, N$ are equivalent

Containment of Queries with Comparisons

Theorem (Klug 88)

If

- Q, Q' are **conjunctive** queries **with comparisons** with set of constants D
- $(Q_N)_N$ is a **linear expansion** of Q over D,

then:

$$Q \sqsubseteq Q' \quad \Leftrightarrow \quad \text{for every } Q_N \text{ in } (Q_N)_N, \text{ there is an homomorphism from } Q' \text{ to } Q_N$$

Corollary

Containment of conjunctive queries with comparisons is in Π_2^P.

Containment of Queries with Comparisons (cntd)

Proof.

Suppose Q', Q, and $(Q_N)_N$ are as in the theorem. Let $W = \textbf{var}(Q)$.

"$\Leftarrow$" If there is a homomorphism from Q' to Q_N, then $Q_N \sqsubseteq Q'$.
Thus, $Q \sqsubseteq Q'$, since $Q \equiv \bigcup_N Q_N$.

"$\Rightarrow$" If $Q \sqsubseteq Q'$, then $Q_N \sqsubseteq Q'$ for every $N \in \mathcal{L}_D(Q)$.
It suffices to show: "$Q_N \sqsubseteq Q' \ \Rightarrow$ there is a homomorphism from Q' to Q_N"

Recall: $Q_N(\phi\bar{x}) \mathrel{:-} \phi L, \phi N$.
Let $\alpha \models N$. N is a linearization of $W \cup D \ \Rightarrow \ \alpha$ is injective on *Terms*(Q_N).
Then: (i) $\mathbf{I}_\alpha = \alpha\phi L$ is an instance, (ii) $\alpha(\phi\bar{x}) \in Q_N(\mathbf{I}_\alpha)$.
Also: $Q_N \sqsubseteq Q' \Rightarrow \alpha(\phi\bar{x}) \in Q'(\mathbf{I}_\alpha)$.
Hence, there is an assignment β' for $\textbf{var}(Q')$ such that

(i) $\mathbf{I}_\alpha, \beta' \models Q'$ and (ii) $\beta'\bar{x} = \alpha\phi\bar{x}$.

Now, due to the injectivity of α on *Terms*(Q_N),

$\beta := \alpha^{-1}\beta'$ is well defined and is a homomorphism from Q' to Q. □

Reminder on the Class PSPACE

PSPACE = the class of problems that can be decided by a deterministic (or nondeterministic) Turing machine with polynomial space

There are PSPACE-complete problems. The best-known PSPACE-complete problem is the one of validity of **quantified Boolean formulas** (QBF).

A quantified Boolean formula (qbf) consists of a **prefix** and a **matrix**:

- the matrix is a propositional formula ϕ
- the prefix is a sequence of quantifications $Q_1 x_1, \ldots, Q_n x_n$ where $x_1, \ldots, x_n$ are the propositions in ϕ and $Q_i \in \{\forall, \exists\}$

An example of a qbf is

$$\forall x\, \exists y\, \exists z\, \forall w\, (x \vee \neg y \vee z) \wedge (y \vee \neg z \vee w)$$

PSPACE-complete Problems

A qbf is **valid** if there is a set of assignments A such that

- Q is compatible with the prefix
- every $\alpha \in A$ satisfies the matrix

Definition (QBF Problem)

Given: a quantified Boolean formula

Question: is the formula valid?

Theorem (PSPACE-Completeness)

The QBF problem is complete for the class PSPACE

What is the combined complexity of the evaluation problem for relational calculus queries? And what is the data complexity?

Reminder on the Polynomial Hierarchy

There are problems in PSPACE that are NP-hard,
but have neither been shown to be in NPnor to be PSPACE-complete.

For a problem P, a Turing machine with a **P-oracle** is an extension of a regular Turing machine that

- can write strings s on a special tape, the oracle tape
- receive a one-step answer whether $s \in P$ or not.

Let $\mathcal{C}$ be a class of problems.

- The class $\mathrm{NP}^{\mathcal{C}}$ consists of all problems that can be solved by a polynomial time nondeterministic Turing machine with an oracle for some $P_0 \in \mathcal{C}$.
- The class $\mathrm{coNP}^{\mathcal{C}}$ consists of all problems P whose complements $\Sigma^* \setminus P$ are in NP^{C}.

Reminder on the Polynomial Hierarchy (cntd)

Definition (Polynomial Hierarchy)

One defines recursively the classes Σ_k^{P}, Π_k^{P} of the **polynomial hierarchy** as

$$\Sigma_0^{\mathsf{P}} = \Pi_0^{\mathsf{P}} = \mathsf{P}$$
$$\Sigma_{k+1}^{\mathsf{P}} = \mathsf{NP}^{\Sigma_k^{\mathsf{P}}}$$
$$\Pi_{k+1}^{\mathsf{P}} = \mathsf{coNP}^{\Sigma_k^{\mathsf{P}}}$$

Note: $\Sigma_1^{\mathsf{P}} = \mathsf{NP}$ and $\Pi_1^{\mathsf{P}} = \mathsf{coNP}$

Complete Problems for the Polynomial Hierarchy

A complete problem for Σ_k^{P} is $\exists\mathsf{QBF}_k$. It consists of all valid qbfs with k alternations of quantifiers, starting with an existential:

$$\exists X_1 \, \forall X_2 \, \ldots, Q_k \, \phi$$

- If k is even, the problem is already complete if ϕ consists of a disjunction of conjunctive 3-clauses.
- If k is odd, the problem is already complete if ϕ consists of a conjunction of disjunctive 3-clauses.

A complete problem for Π_k^{P} is $\forall\exists\mathsf{QBF}_k$. It consists of all valid qbfs with k alternations of quantifiers, starting with a universal:

$$\forall X_1 \, \exists X_2 \, \ldots, Q_k \, \phi$$

Analogous subclasses to the ones above are already complete for Π_k^{P}.

In particular, $\forall\exists$3SAT is complete for Π_2^{P}

Containment of Queries with Comparisons

Theorem (van der Meyden 92)

Containment with comparisons is Π_2^P-complete.

The proof here is different from the one by van der Meyden.

It uses a simple pattern that can be used to prove many more Π_2^P-hardness results about query containment, for instance, containment of queries

- with the predicate "$\neq$"
- with negated subgoals (like $\neg R(x)$)
- SQL null values.

Reduction of ∀∃3SAT to Containment with Comparisons

We show the reduction for a general formula

$$\psi = \forall x_1, \ldots, x_m \exists y_1, \ldots, y_n \, \gamma_1 \wedge \ldots \wedge \gamma_k$$

where $\gamma_1, \ldots, \gamma_k$ are disjunctive 3-clauses, and for the example

$$\psi_0 \;=\; \forall x_1 \, \forall x_2 \, \exists y_1 \, \exists y_2 \, (x_1 \vee \neg x_2 \vee y_1) \wedge (x_2 \vee \neg y_1 \vee y_2)$$

We define boolean queries Q', Q such that $Q \sqsubseteq Q'$ iff ψ is valid.

Reduction of ∀∃3SAT to Containment with Comparisons

We model the **universal quantifiers** $\forall x_i$
by pairs of "generator conditions" G_i', G_i,
following the "black and white blocks" example:

$$G_i' = S_i(u_i, v_i, x_i), u_i \leq 4, v_i > 4$$

$$G_i = S_i(4, w_i, 1), S_i(w_i, 5, 0)$$

Idea: For G_i' to be more general than G_i

- x_i must be mapped to 1, if w_i is bound to a value ≤ 4
- x_i must be mapped to 0, otherwise.

Reduction of ∀∃3SAT to Containment with Comparisons

For every clause γ_i, we introduce

$$H'_i = R_i(p_1^{(i)}, p_2^{(i)}, p_3^{(i)})$$
$$H_i = R_i(\bar{t}_1^{(i)}), \ldots, R_i(\bar{t}_7^{(i)})$$

where $p_1^{(i)}$, $p_2^{(i)}$, $p_3^{(i)}$ are the three propositions occuring in γ_i and $\bar{t}_1^{(i)}, \ldots, \bar{t}_7^{(i)}$ are the seven combinations of truth values that satisfy γ_i.

In our example

$$\begin{aligned} H'_1 &= R_1(x_1, x_2, y_1) \\ H_1 &= R_1(0,0,0),\ R_1(0,0,0),\ R_1(0,1,1), \\ &\quad\ R_1(1,0,0),\ R_1(1,0,1),\ R_1(1,1,0),\ R_1(1,1,1) \end{aligned}$$

Reduction of ∀∃3SAT to Containment with Comparisons

The queries for ψ are

$$Q'() \coloneq G'_1, \ldots, G'_m, H'_1, \ldots, H'_n$$
$$Q() \coloneq G_1, \ldots, G_m, H_1, \ldots, H_n$$

Lemma

$$Q \sqsubseteq Q' \qquad \text{iff} \qquad \psi \text{ is valid}$$

Sketch.

"$\Leftarrow$" For cach binding of the w_i in Q over a db instance, we can map G'_i to one of the atoms in G_i. Such a mapping corresponds to a choice of 0 or 1 for x_i. If ψ is valid, then for every binding of the x_i we find values for the y_j that satisfy all clauses. These values allows us to map H'_l to one of the atoms in H_i

"$\Rightarrow$" For each assignment of 0, 1 to the x_i, we create a db instance by instantiating w_i in Q with 4 or 5. This instance satisfies Q. It must also satisfy Q'. This tells us that we can instantiate the y_j such that ψ is satisfied. □

Minimizing Conjunctive Queries

- A conjunctive query may have atoms that can be dropped without changing the answers.
- Since computing joins is expensive, this has the potential of saving computation cost

Goal: Given a conjunctive query Q, find an equivalent conjunctive query Q' with the minimum number of joins.

Questions: How many such queries can exist?

How different are they?

How can we find them?

Assumption: We consider only relational CQs.

The "Drop Atoms" Algorithm

Input: $Q(\bar{x})$:– L

$L' := L$;
repeat until no change
  choose an atom $A \in L$;
  if there is a homomorphism
   from $Q(\bar{x})$:– L' to $Q(\bar{x})$:– $L' \setminus \{A\}$
  then $L' := L' \setminus \{A\}$
end

Output: $Q'(\bar{x})$:– L'

Questions About the Algorithm

- Does it terminate?
- Is Q' equivalent to Q?
- Is Q' of minimal length among the queries equivalent to Q?

Subqueries

Definition (Subquery)

If Q is a conjunctive query,

$$Q(\bar{x}) \mathrel{:-} R_1(\bar{t}_1), \ldots, R_k(\bar{t}_k),$$

then Q' is a **subquery** of Q if Q' is of the form

$$Q'(\bar{x}) \mathrel{:-} R_{i_1}(\bar{t}_{i_1}), \ldots, R_{i_l}(\bar{t}_{i_l})$$

where $1 \leq i_1 < i_2 < \ldots < i_l \leq k$.

Proposition

The Drop-Atoms Algorithm outputs a subquery Q' of Q such that

- Q' and Q are equivalent
- Q' does not have a subquery equivalent to Q.

To Minimize Q, It's Enough To Shorten Q

Proposition

Consider the relational conjunctive query

$$Q(\bar{x}) \coloneq R_1(\bar{t}_1), \ldots, R_n(\bar{t}_n).$$

If there is an equivalent conjunctive query

$$Q'(\bar{x}) \coloneq S_1(\bar{s}_1), \ldots, S_l(\bar{s}_m), \qquad m < k,$$

then Q_0 is equivalent to a subquery

$$Q_0(\bar{x}) \coloneq R_{i_1}(\bar{t}_{i_1}), \ldots, R_{i_l}(\bar{t}_{i_l}), \qquad l \leq m.$$

In other words: If Q is a relational CQ with n atoms and Q' an equivalent relational CQ with m atoms, where $m < n$, then there exists a subquery Q_0 of Q such that Q_0 has at most m atoms in the body and Q_0 is equivalent to Q.

Proof as exercise!

unibz.it

Minimization Theorem

Theorem (Minimization)

Let Q and Q' be two equivalent minimal relational CQs. Then Q and Q' are identical up to renaming of variables.

Proof as exercise!

Conclusions:

- There is essentially one minimal version of each relational CQ Q
- We can obtain it by dropping atoms from Q's body
- The Drop-Atoms algorithm is sound and complete

Minimizing SPJ/Conjunctive Queries: Example

Consider relation R with three attributes A, B, C and the SPJ query

$$Q \quad = \quad \pi_{AB}(\sigma_{B=4}(R)) \bowtie \pi_{BC}(\pi_{AB}(R) \bowtie \pi_{AC}(\sigma_{B=4}(R)))$$

- Translate into relational calculus:

$$(\exists z_1 \ R(x, y, z_1) \wedge y = 4) \ \wedge \ \exists x_1 \ \Big((\exists z_2 \ R(x_1, y, z_2)) \ \wedge \ (\exists y_1 \ R(x_1, y_1, z) \wedge y_1 = 4) \Big)$$

- Simplify, by substituting the constant, and pulling quantifiers outward:

$$\exists x_1, z_1, z_2 \, (R(x, 4, z_1) \wedge R(x_1, 4, z_2) \wedge R(x_1, 4, z) \wedge y = 4)$$

- Conjunctive query:

$$Q(x, y, z) \coloneq R(x, 4, z_1),\ R(x_1, 4, z_2),\ R(x_1, 4, z),\ y = 4$$

Then minimize: Exercise!

Minimization of Queries with Built-Ins

For queries with built-ins, things become more difficult:

Example (Gottlob)

$$
\begin{aligned}
Q() \mathrel{:-} & R(x_1, x_2),\ R(x_2, x_3),\ R(x_3, x_4),\ R(x_4, x_5),\ R(x_5, x_1), \\
& x_1 \neq x_2 \\
Q'() \mathrel{:-} & R(x_1, x_2),\ R(x_2, x_3),\ R(x_3, x_4),\ R(x_4, x_5),\ R(x_5, x_1), \\
& x_1 \neq x_3
\end{aligned}
$$

We note

- Q, Q' are equivalent
  (assume they are not, and find a contradiction!)
- there is no homomorphism $Q \to Q'$ and no homomorphism $Q' \to Q$

Minimization of Queries with Built-Ins (Cntd)

There is no theory yet about minimization of CQs with Built-Ins.

To the best of my knowledge, the following questions are still open:

- Are there CQs Q, Q' with comparisons that are equivalent, but cannot be mapped homomorphically to each other?
- Are there CQs Q, Q' with built-ins that are equivalent, but have different numbers of atoms?
- How similar are the results of the Drop-Atoms Algorithm, if we apply it to CQs with built-ins?

Functional Dependencies

Consider the relation

$$\texttt{Lect}(\texttt{name}, \texttt{office}, \texttt{course})$$

For any university instance,

- all tuples with the same "`name`" have the same "`office`" value
- tuples may have the same "`course`", but different "`name`" and "`office`"
  (if lecturers share courses)
- tuples may have the same "`office`", but different "`name`" and "`course`"
  (if lecturers share offices)

Functional Dependencies (Cntd)

$$\texttt{Lect}(\texttt{name}, \texttt{office}, \texttt{course})$$

The formula

$$\forall n,\, o_1,\, c_1,\, o_2,\, c_2\, (\texttt{Lect}(n, o_1, c_1) \land \texttt{Lect}(n, o_2, c_2) \ \rightarrow\ o_1 = o_2)$$

is a **functional dependency** (FD).

Assuming that `Lect` is clear from the context, we abbreviate it as

$$\texttt{name} \ \rightarrow\ \texttt{office}$$

and read "`name` determines `office`".

FDs are a frequent type of integrity constraints (keys are a special case)

Functional Dependencies (Cntd)

Notation:

- If R is relation with attribute set Z, we write FDs as

$$X \to A \quad \text{or} \quad X \to Y$$

  where X, $Y \subseteq Z$ and $A \in Z$
- X, Y, Z represent sets of attributes; A, B, C represent single attributes
- no set braces in sets of attributes: just ABC, rather than $\{A, B, C\}$

Semantics:

- $X \to Y$ is satisfied by an instance $\mathbf{I}$, that is $\mathbf{I} \models X \to Y$, iff

$$\pi_X(t) = \pi_X(t') \text{ implies } \pi_Y(t) = \pi_Y(t'), \text{ for all } t, t' \in \mathbf{I}(R)$$

- Note: $X \to AB$ is a equivalent to $X \to A$ and $X \to B$

  $\Rightarrow$ it suffices to deal with FDs $X \to A$

Equivalence wrt Functional Dependencies

Consider the queries

$$Q = \texttt{Lect}$$

$$Q' = \pi_{\texttt{name},\texttt{course}}(\texttt{Lect}) \bowtie_{\texttt{name}} \pi_{\texttt{name},\texttt{office}}(\texttt{Lect})$$

- In general, is there equivalence/containment among Q, Q'?
- What if we take into account the FD $\texttt{name} \rightarrow \texttt{office}$?

Instead of algebra, let's use rule notation

$$Q(n, o, c) \text{ :- } \texttt{Lect}(n, o, c)$$
$$Q'(n, o, c) \text{ :- } \texttt{Lect}(n, o', c), \texttt{Lect}(n, o, c')$$

Chase and Miminize

$$Q'(n, o, c) \mathrel{:\!-} \texttt{Lect}(n, o', c), \texttt{Lect}(n, o, c')$$

Using the FD `name` $\rightarrow$ `office`, we infer $o = o'$:

$$Q'(n, o, c) \mathrel{:\!-} \texttt{Lect}(n, o, c), \texttt{Lect}(n, o, c')$$

Minimizing using Drop Atom, we get

$$Q'(n, o, c) \mathrel{:\!-} \texttt{Lect}(n, o, c)$$

Thus, $Q' \equiv Q$

FD Violations

Notation: Instead of $\pi_X(t)$ and $\pi_A(t)$, we write $t.X$ and $t.A$

Definition (Violation)

The FD $X \rightarrow A$ over R is **violated** by the atoms $R(t)$, $R(t')$ if

- $t.X = t'.X$ and
- $t.A \neq t'.A$

The Chase Algorithm

Input: query $Q(\bar{s})$:– L, set of FDs $\mathcal{F}$

let $(\bar{s}', L') = (s, L)$
while
L' contains atoms $R(t)$, $R(t')$,
violating some $X \rightarrow A \in \mathcal{F}$ **do**
case $t.A$, $t'.A$ **of**
- one is a nondistinguished variable
  $\Rightarrow$ in $(\bar{s}', L')$, replace the nondistinguished variable by the other term
- one is a distinguished variable,
  the other one a distinguished variable or constant
  $\Rightarrow$ in $(\bar{s}', L')$, replace the distinguished variable by the other term
- both are constants
  $\Rightarrow$ set $L' = \perp$ and **stop**

end
end

Output: query $Q'(\bar{s}')$:– L'

Questions about the Chase Algorithm

- Does the Chase algorithm terminate? What is the run time?
- What is the relation between a query and its Chase'd version?
- Query containment wrt a set of FDs:
  - How can we define this problem?
  - Can we decide this problem?
- Query minimization wrt to a set of FDs:
  - How can we define this problem?
  - How can we solve it?
- Relational CQs:
  - We know that all such queries are satisfiable.
    Is this still true if we allow only instances
    that satisfy a given set of FDs?