Relational Databases, Logic, and Complexity

Phokion G. Kolaitis

University of California, Santa Cruz

&

IBM Research-Almaden

kolaitis@cs.ucsc.edu

2009 GII Doctoral School on Advances in Databases

What this course is about

Goals:

- Cover a coherent body of basic material in the foundations of relational databases
- Prepare students for further study and research in relational database systems

Overview of Topics:

- Database query languages: expressive power and complexity
  - Relational Algebra and Relational Calculus
  - Conjunctive queries and homomorphisms
  - Recursive queries and Datalog
- Selected additional topics: Bag Semantics, Inconsistent Databases

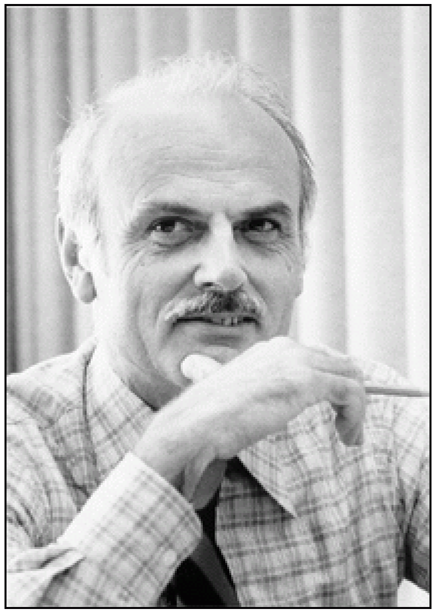
Unifying Theme:

The interplay between databases, logic, and computational complexity

Relational Databases: A Very Brief History

- The history of relational databases is the history of a scientific and technological revolution.
- The scientific revolution started in 1970 by Edgar (Ted) F. Codd at the IBM San Jose Research Laboratory (now the IBM Almaden Research Center)
- Codd introduced the relational data model and two database query languages: relational algebra and relational calculus.
  - "A relational model for data for large shared data banks", CACM, 1970.
  - "Relational completeness of data base sublanguages", in: Database Systems, ed. by R. Rustin, 1972.

Edgar F. Codd, 1923-2003



Relational Databases: A Very Brief History

- Researchers at the IBM San Jose Laboratory embark on the System R project, the first implementation of a relational database management system (RDBMS)
  - In 1974-1975, they develop SEQUEL, a query language that eventually became the industry standard SQL.
  - System R evolved to DB2 – released first in 1983.
- M. Stonebraker and E. Wong embark on the development of the Ingres RDBMS at UC Berkeley in 1973.
  - Ingres is commercialized in 1983; later, it became PostgreSQL, a free software OODBMS (object-oriented DBMS).
- L. Ellison founds a company in 1979 that eventually becomes Oracle Corporation; Oracle V2 is released in 1979 and Oracle V3 in 1983.
- Ted Codd receives the ACM Turing Award in 1981.

Relational Database Industry Today

- According to Gartner, Inc., June 2007:

  "Worldwide relational database management systems (RDBMS) total software revenue totaled $15.2 billion in 2006, a 14.2 percent increase from 2005 revenue of $13.3 billion."

- In 2007, the total RDBMS software revenue increased to $17.1 billion (figures released in July 2008).

Company	2006 Revenue	2006 Market Share
Oracle	7.168B	47.1%
IBM	3.204B	21.1%
Microsoft	2.654B	17.4%
Teradata	494.2M	3.2%
Sybase	486.7M	3.2%
Other	1.2B	7.8%
Total	15.2B	100%

Database Research Today

- A very vibrant community comprising several thousand researchers around the world.
- Several major annual conferences in database research:
  - SIGMOD, PODS, VLDB, ICDE, EDBT, ICDT (top six).
  - Numerous other conferences and workshops.
- Several major scholarly journals dedicated to database research:
  - ACM TODS, VLDB Journal, IEEE TKDE, …
- Strong database research groups in academia around the world.
- Several database research groups in industrial laboratories.

Database Management Systems

A database management system (DBMS) provides support for:

- At least one data model (a mathematical abstraction for representing data);
- At least one high level data language (language for defining, updating, manipulating, and retrieving data);
- Transaction management & concurrency control mechanisms;
- Access control (limit access of certain data to certain users);
- Resiliency (ability to recover from crashes).

Data Models and Data Languages

- A data model is a mathematical formalism for describing and representing data.
- A data model is accompanied by a data language that has two parts: a data definition language and a data manipulation language.
  - A data definition language (DDL) has a syntax for describing "database templates" in terms of the underlying data model.
  - A data manipulation language (DML) supports the following operations on data:
    - Insertion
    - Deletion
    - Update
    - Retrieval and extraction of data (query the data).
  - The first three operations are fairly standard. However, there is much variety on data retrieval and extraction (query languages).

The Relational Data Model (E.F. Codd – 1970)

- The Relational Data Model uses the mathematical concept of a relation as the formalism for describing and representing data.
- Question: What is a relation?
- Answer:
    - Formally, a relation is a subset of a cartesian product of sets.
    - Informally, a relation is a "table" with rows and columns.

The Relational Data Model (E.F. Codd – 1970)

- The Relational Data Model uses the mathematical concept of a relation as the formalism for describing and representing data.
- Question: What is a relation?
- Answer:
  - Formally, a relation is a subset of a cartesian product of sets.
  - Informally, a relation is a "table" with rows and columns.

CHECKING-ACCOUNT Table

branch-name	**account-no**	**customer-name**	**balance**
Orsay	10991-06284	Abiteboul	$3,567.53
Hawthorne	10992-35671	Hull	$11,245.75
…	…	…	…

Basic Notions from Discrete Mathematics

- A k-tuple is an ordered sequence of k objects (need not be distinct)
  - (2,0,1) is a 3-tuple; (a,b,a,a,c) is a 5-tuple, and so on.

- If $D_1, D_2, \ldots, D_k$ are k sets, then the cartesian product $D_1 \times D_2 \ldots \times D_k$ of these sets is the set of all k-tuples $(d_1,d_2, \ldots,d_k)$ such that $d_i \in D_i$, for $1 \leq i \leq k$.

- Fact: Let $|D|$ denote the cardinality (= # of elements) of a set D. Then $|D_1 \times D_2 \times \ldots \times D_k| = |D_1| \times |D_2| \times \ldots \times |D_k|$.

- Example: If $D_1 = \{0,1\}$ and $D_2 = \{a,b,c,d\}$, then $|D_1| \times |D_2| = 8$.

- Warning: Computing cartesian products is an expensive operation!

Basic Notions from Discrete Mathematics

- A k-ary relation R is a subset of a cartesian product of k sets, i.e.,
  - $R \subseteq D_1 \times D_2 \times \ldots \times D_k$.
- Examples:
  - Unary $R = \{0,2,4,\ldots,100\}$ $(R \subseteq D)$
  - Binary $T = \{(a,b)$: a and b have the same birthday$\}$
  - Ternary $S = \{(m,n,s): s = m+n\}$
  - …

Relations and Attributes

- Note:

  $R \subseteq D_1 \times D_2 \times \ldots \times D_k$ can be viewed as a table with k columns

  Table S

3	5	8
150	100	250
…	…	…

- In the relational data model, we want to have names for the columns; these are the attributes of the relation.

Relation Schemas and Relational Database Schemas

- A k-ary relation schema **R**$(A_1,A_2,\ldots,A_K)$ is a set $\{A_1,A_2,\ldots,A_k\}$ of k attributes.
  - **COURSE**(course-no, course-name, term, instructor, room, time)
  - **CITY-INFO**(name, state, population)

  Thus, a k-ary relation schema is a "blueprint", a "template" for some k-ary relation.

- An instance of a relation schema is a relation conforming to the schema (arities match; also, in DBMS, data types of attributes match).

- A relational database schema is a set of relation schemas $\mathbf{R_i}(A_1,A_2,\ldots,A_{k_i})$, for $1\leq i\leq m$.

- A relational database instance of a relational schema is a set of relations R_i each of which is an instance of the relation schema $\mathbf{R_i}$, $1\leq i\leq m$.

Relational Database Schemas - Examples

- BANKING relational database schema with relation schemas
  - CHECKING-ACCOUNT(branch, acc-no, cust-id, balance)
  - SAVINGS-ACCOUNT(branch, acc-no, cust-id, balance)
  - CUSTOMER(cust-id, name, address, phone, email)
  - ....
- UNIVERSITY relational database schema with relation schemas
  - STUDENT(student-id, student-name, major, status)
  - FACULTY(faculty-id, faculty-name, dpt, title, salary)
  - COURSE(course-no, course-name, term, instructor)
  - ENROLLS(student-id, course-no, term)
  - ...

Schemas vs. Instances

Keep in mind that there is a clear distinction between

- relation schemas and instances of relation schemas

and between

- relational database schemas and relational database instances.

Syntactic Notion	**Semantic Notion (discrete mathematics notion)**
Relation Schema	Instance of a relation schema (i.e., a relation)
Relational Database Schema	Relational database instance (i.e., a database)

Programming Languages Paradigms

There are two main paradigms of programming languages: imperative (or procedural) languages and declarative languages.

- Imperative (Procedural) Languages: programs are expressed by specifying how the task is to be accomplished (sequence of operations).
  - FORTRAN, C, …
- Declarative Languages: programs are expressed by specifying what has to be accomplished (as opposed to "how").
  - LISP (functional programming), PROLOG (logic programming), …

Query Languages for the Relational Data Model

Codd introduced two different query languages for the relational data model:

- Relational Algebra, which is a procedural language.
  - It is an algebraic formalism in which queries are expressed by applying a sequence of operations to relations.
- Relational Calculus, which is a declarative language.
  - It is a logical formalism in which queries are expressed as formulas of first-order logic.

Codd's Theorem: Relational Algebra and Relational Calculus are essentially equivalent in terms of expressive power.

(but what does this really mean?)

Desiderata for a Database Query Language

Desiderata:

- The language should be sufficiently high-level to secure physical data independence, i.e., the separation between the physical level and the conceptual level of databases.
- The language should have high enough expressive power to be able to pose useful and interesting queries against the database.
- The language should be efficiently implementable to allow for the fast retrieval of information from the database.

Warning:

- There is a tension between the last two desiderata.
- Increase in expressive power comes at the expense of efficiency.

Relational Algebra

- Relational algebra strikes a good balance between expressive power and efficiency.
- Codd's key contribution was to identify a small set of basic operations on relations and to demonstrate that useful and interesting queries can be expressed by combining these operations.
  - Thus, relational algebra is a rich enough language, even though, as we will see later on, it suffers from certain limitations in terms of expressive power.
- The first RDBMS prototype implementations (System R and Ingres) demonstrated that the relational algebra operations can be implemented efficiently.

The Five Basic Operations of Relational Algebra

- Group I: Three standard set-theoretic binary operations:
  - Union
  - Difference
  - Cartesian Product.
- Group II. Two special unary operations on relations:
  - Projection
  - Selection.
- Relational Algebra consists of all expressions obtained by combining these five basic operations in syntactically correct ways.

Relational Algebra: Standard Set-Theoretic Operations

- Union
  - Input: Two k-ary relations R and S, for some k.
  - Output: The k-ary relation $R \cup S$, where
    $R \cup S = \{(a_1,\ldots,a_k): (a_1,\ldots,a_k) \text{ is in } R \text{ or } (a_1,\ldots,a_k) \text{ is in } S\}$
- Difference:
  - Input: Two k-ary relations R and S, for some k.
  - Output: The k-ary relation $R - S$, where
    $R - S = \{(a_1,\ldots,a_k): (a_1,\ldots,a_k) \text{ is in } R \text{ and } (a_1,\ldots,a_k) \text{ is not in } S\}$
- Note:
  - In relational algebra, both arguments to the union and the difference must be relations of the same arity.
  - In SQL, there is the additional requirement that the corresponding attributes must have the same data type.

Relational Algebra: Standard Set-Theoretic Operations

- Cartesian Product
  - Input: An m-ary relation R and an n-ary relation S
  - Output: The (m+n)-ary relation $R \times S$, where

$$R \times S = \{(a_1,\ldots,a_m,b_1,\ldots,b_n): (a_1,\ldots a_m) \text{ is in } R \text{ and } (b_1,\ldots,b_n) \text{ is in } S\}$$

- Note: As stated earlier,

$$|R\times S| = |R| \times |S|$$

The Projection Operator

- Motivation:

  It is often the case that, given a table R, one wants to:
  - Rearrange the order of the columns
  - Suppress some columns
  - Do both of the above.

- Fact: The Projection Operation is tailored for this task

The Projection Operation

- Projection is a family of unary operations of the form

  $\pi_{\text{<attribute list>}}$ (<relation name>)

- The intuitive description of the projection operation is as follows:
  - When projection is applied to a relation R, it removes all columns whose attributes do not appear in the <attribute list>.
  - The remaining columns may be re-arranged according to the order in the <attribute list>.
  - Any duplicate rows are also eliminated.

The Projection Operation - Example

SAVINGS

branch-name	acc-no	cust-name	balance
Aptos	153125	Vianu	3,450
Santa Cruz	123658	Hull	2,817
San Jose	321456	Codd	9,234
San Jose	334789	Codd	875

$\pi_{\text{cust-name,branch-name}}(\text{SAVINGS})$

cust-name	branch-name
Vianu	Aptos
Hull	Santa Cruz
Codd	San Jose

More on the Syntax of the Projection Operation

- In relational algebra, attributes can be referenced by position no.
- Projection Operation:
  - Syntax: $\pi_{i_1,\ldots,i_m}(R)$, where R is of arity k, and i_1, ….i_m are distinct integers from 1 up to k.
  - Semantics:

$$\pi_{i_1,\ldots,i_m}(R) = \{(a_1,\ldots,a_m)\text{: there is a tuple } (b_1,\ldots,b_k) \text{ in } R \text{ such that } a_1 = b_{i_1}, \ldots, a_m = b_{i_m}\}$$

- Example: If R is R(A,B,C,D), then $\pi_{C,A}(R) = \pi_{3,1}(R)$

The Selection Operation

- Motivation:
  - Consider the table

    SAVINGS(branch-name, acc-no, cust-name, balance)
  - We may want to extract the following information from it:
    - Find all records in the Aptos branch
    - Find all records with balance at least $50,000
    - Find all records in the Aptos branch with balance less than $1,000
- Fact: The Selection Operation is tailored for this task.

The Selection Operation

- Selection is a family of unary operations of the form

$$\sigma_{\Theta}\ (\mathrm{R}),$$

  where R is a relation and Θ is a condition that can be applied as a test to each row of R.

- When a selection operation is applied to R, it returns the subset of R consisting of all rows that satisfy the condition Θ

- Question: What is the precise definition of a “condition”?

The Selection Operation

- Definition: A condition in the selection operation is an expression built up from:
  - Comparison operators =, <, >, ≠, ≤, ≥ applied to operands that are constants or attribute names or component numbers.
    - These are the basic (atomic) clauses of the conditions.
  - The Boolean logic operators ∧, ∨, ¬ applied to basic clauses.
- Examples:
  - balance > 10,000
  - branch-name = "Aptos"
  - (branch-name = "Aptos") ∧ (balance < 1,000)

The Selection Operator

- Note:
  - The use of the comparison operators <, >, ≤, ≥ assumes that the underlying domain of values is totally ordered.
  - If the domain is not totally ordered, then only = and ≠ are allowed.
  - If we do not have attribute names (hence, we can only reference columns via their component number), then we need to have a special symbol, say $, in front of a component number. Thus,
    - $4 > 100 is a meaningful basic clause
    - $1 = "Aptos" is a meaningful basic clause, and so on.

Relational Algebra

- Definition: A relational algebra expression is a string obtained from relation schemas using union, difference, cartesian product, projection, and selection.

- Context-free grammar for relational algebra expressions:

  E := R, S, … | ($E_1 \vee E_2$) | ($E_1 - E_2$) | ($E_1 \times E_2$) | π_L(E) | σ_Θ(E),

  where
  - R, S, … are relation schemas
  - L is a list of attributes
  - Θ is a condition.

Strength from Unity and Combination

- By itself, each basic relational algebra operation has limited expressive power, as it carries out a specific and rather simple task.
- When used in combination, however, the five relational algebra operations can express interesting and, quite often, rather complex queries.
- Derived relational algebra operations are operations on relations that are expressible via a relational algebra expression (built from the five basic operators).

Intersection

- Intersection
  - Input: Two k-ary relations R and S, for some k.
  - Output: The k-ary relation $R \cap S$, where

$$R \cap S = \{(a_1,\ldots,a_k) : (a_1,\ldots,a_k) \text{ is in } R \text{ and } (a_1,\ldots,a_k) \text{ is in } S\}$$

- Fact: $R \cap S = R - (R - S) = S - (S - R)$

Thus, intersection is a derived relational algebra operation.

Natural Join

- Fact: The most FAQs against databases involve the natural join operation $\bowtie$.

- Motivating Example: Given
  TEACHES(fac-name,course,term) and
  ENROLLS(stud-name,course,term),

  we want to obtain
  TAUGHT-BY(stud-name,course,term,fac-name)

  It turns out that TAUGHT-BY = ENROLSS $\bowtie$ TEACHES

Natural Join

Given TEACHES(fac-name,course,term) and

ENROLLS(stud-name, course,term):

To compute TAUGHT-BY(stud-name,course,term,fac-name)

1. ENROLLS $\times$ TEACHES
2. $\sigma_{\text{T.course = E.course} \wedge \text{T.term = E.term}}$ (ENROLLS $\times$ TEACHES)
3. $\pi_{\text{stud-name,E.course,E.term,fac-name}}$ ($\sigma_{\text{T.course = E.course} \wedge \text{T.term = E.term}}$ (ENROLLS $\times$ TEACHES))

The result is ENROLLS $\bowtie$ TEACHES.

Natural Join

- **Definition:** Let A1, …, Ak be the common attributes of two relation schemas R and S. Then

$$R \bowtie S = \pi_{<list>} (\sigma_{R.A1=S.A1 \wedge \ldots \wedge R.A1 = S.Ak} (R \times S)),$$

where <list> contains all attributes of $R \times S$, except for

S.A1, …, S.Ak (in other words, duplicate columns are eliminated).

- **Algorithm for $R \bowtie S$:**

  For every tuple in R, compare it with every tuple in S as follows:
  - test if they agree on all common attributes of R and S;
  - if they do, take the tuple in $R \times S$ formed by these two tuples,
  - remove all values of attributes of S that also occur in R;
  - put the resulting tuple in $R \bowtie S$.

Quotient (Division)

- Motivating Example:
  Given ENROLLS(stud-name,course) and TEACHES(fac-name,course), find the names of students who take every course taught by V. Vianu.

- Other Motivating Examples:
  - Find the names of customers who have an account in every branch of Wachovia in San Jose.
  - Find the names of Netflix customers who have rented every film in which Paul Newman starred.

- These and other similar queries can be answered using the Quotient (Division) operation.

Quotient (Division)

- Definition: Let R be a relation of arity r and let S be a relation of arity s, where r > s.

  The quotient (or division) R ÷ S is the relation of arity r – s consisting of all tuples $(a_1,\ldots,a_{r-s})$ such that for every tuple $(b_1,\ldots,b_s)$ in S, we have that $(a_1,\ldots,a_{r-s}, b_1,\ldots,b_s)$ is in R.

- Example: Given

  ENROLLS(stud-name,course) and TEACHES(fac-name,course), find the names of students who take every course taught by V. Vianu.

  - Find the courses taught by V. Vianu

    $$\pi_{\text{course}}\,(\sigma_{\text{fac-name = “V. Vianu”}}\,(\text{TEACHES}))$$

  - The desired answer is given by the expression:

    $$\text{ENROLLS} \div \pi_{\text{course}}\,(\sigma_{\text{fac-name = “V. Vianu”}}\,(\text{TEACHES}))$$

Quotient (Division)

Fact: The quotient operation is expressible in relational algebra.

Proof: For concreteness, assume that R has arity 5 and S has arity 2.

Key Idea: Use the difference operation

- $R \div S = \pi_{1,2,3}(R)$ – "tuples in $\pi_{1,2,3}(R)$ that do not make it to $R \div S$"
- Consider the relational algebra expression $(\pi_{1,2,3}(R) \times S) - R$.

  Intuitively, it is the set of all tuples that fail the test for membership in $R \div S$. Hence,

- $R \div S = \pi_{1,2,3}(R) - \pi_{1,2,3}((\pi_{1,2,3}(R) \times S) - R)$.

The Expressive Power of Relational Algebra

- When combined together, the five basic relational algebra operations can express interesting and complex queries.

- In particular, relational algebra can express:
  - The Intersection Operation
  - The Natural Join Operation
  - The Quotient Operation
  - ....

Independence of the Basic Relational Algebra Operations

- Question: Are all five basic relational algebra operations really needed? Can one of them be expressed in terms of the other four?

- Theorem: Each of the five basic relational algebra operations is independent of the other four, that is, it cannot be expressed by a relational algebra expression that involves only the other four.

  Proof Idea: For each relational algebra operation, we need to discover a property that is possessed by that operation, but is not possessed by any relational algebra expression that involves only the other four operations.

Independence of the Basic Relational Algebra Operations

Theorem: Each of the five basic relational algebra operations is independent of the other four, that is, it cannot be expressed by a relational algebra expression that involves only the other four.

Proof Sketch: (projection and cartesian product only)

- Property of projection:
  - It is the only operation whose output may have arity smaller than its input.
  - Show, by induction, that the output of every relational algebra expression in the other four basic relational algebra is of arity at least as big as the maximum arity of its arguments.

- Property of cartesian product:
  - It is the only operation whose output has arity bigger than its input.
  - Show, by induction, that the output of every relational algebra expression in the other four basic relational algebra is of arity at most as big as the maximum arity of its arguments.

Exercise: Complete this proof.

Relational Algebra: Summary

- When combined with each other, the five basic relational algebra operations can express interesting and complex queries (natural join, quotient, …)
- The five basic relational algebra operations are independent of each other: none can be expressed in terms of the other four.
- So, in conclusion, Codd’s choice of the five basic relational algebra operations has been very judicious.

Relational Completeness

- Definition (Codd – 1972): A database query language L is relationally complete if it is at least as expressive as relational algebra, i.e., every relational algebra expression E has an equivalent expression F in L.
- Relational completeness provides a benchmark for the expressive power of a database query language.
- Every commercial database query language should be at least as expressive as relational algebra.
- Exercise: Explain why SQL is relationally complete.

SQL vs. Relational Algebra

SQL	**Relational Algebra**
SELECT	Projection π
FROM	Cartesian Product $\times$
WHERE	Selection σ

Semantics of SQL via interpretation to Relational Algebra

SELECT R_{i1}.A1, …, R_{im}.A.m
FROM R_1, …,R_K = $\pi_{\text{Ri1.A1, …, Rim.A.m}} (\sigma_{\Psi} (R_1 \times \ldots \times R_K))$
WHERE Ψ

Relational Calculus

- In addition to relational algebra, Codd introduced relational calculus.
- Relational calculus is a declarative database query language based on first-order logic.
- Relational calculus comes into two different flavors:
  - Tuple relational calculus
  - Domain relational calculus.

  We will focus on domain relational calculus.

  There is an easy translation between these two formalisms.
- Codd's main technical result is that relational algebra and relational calculus have essentially the same expressive power.

Propositional Logic (aka Boolean Logic) Reminder

- Propositional variables: x, y, z, …
  - They take values 0 (True) and 1 (False).

- Propositional connectives: $\wedge$, $\vee$, $\neg$, $\rightarrow$

- Propositional formulas: expressions built from propositional variables and propositional connectives
  - Syntax: φ := x, y, z, … | $(\psi \wedge \chi)$ | $(\psi \vee \chi)$ | $\neg\,\psi$ | $(\psi \rightarrow \chi)$
  - Semantics: Truth-table semantics

- Application: Propositional formulas express Boolean functions
  - $(x \vee y) \wedge (\neg\, x \vee \neg\, y)$ XOR-Gate
  - $(x \wedge y) \vee (x \wedge z) \vee (y \wedge z)$ Majority Gate

First-Order Logic - Motivation

- First-Order Logic is a formalism for expressing properties of mathematical structures (graphs, trees, partial orders, …).

- Example: Consider a graph G=(V,E) (nodes are in V, edges are in E)
  - There is a self-loop.
  - Every two nodes are connected via a path of length 2.
  - Every node has exactly three distinct neighbors.
  - There is a path of length 3 from node x to node y.
  - Node x has at least four distinct neighbors

  These and many other similar properties are expressible as formulas of first-order logic on graphs.

- One of Codd’s key insights was that first-order logic can also be used to express relational database queries.

First-Order Logic

- Question: What is First-Order Logic?
- Answer: Informally,

  " First-Order Logic = Propositional Logic + (∃ and ∀)",

  where

  ∃ and ∀ range over possible values occurring in relations.

Relational Calculus (First-Order Logic for Databases)

- First-order variables: $x, y, z, \ldots, x_1, \ldots, x_k, \ldots$
  - They range over values that may occur in tables.
- Relation symbols: R, S, T, … of specified arities (names of relations)
- Atomic (Basic) Formulas:
  - $R(x_1,\ldots,x_k)$, where R is a k-ary relation symbol (alternatively, $(x_1,\ldots,x_k) \in R$; the variables need not be distinct)
  - $(x \text{ op } y)$, where op is one of $=, \neq, <, >, \leq, \geq$
  - $(x \text{ op } c)$, where c is a constant and op is one of $=, \neq, <, >, \leq, \geq$.
- Relational Calculus Formulas:
  - Every atomic formula is a relational calculus formula.
  - If φ and ψ are relational calculus formulas, then so are:
    - $(\varphi \wedge \psi)$, $(\varphi \vee \psi)$, $\neg \psi$, $(\varphi \rightarrow \psi)$ (propositional connectives)
    - $(\exists x \, \varphi)$ (existential quantification)
    - $(\forall x \, \varphi)$ (universal quantification).

Relational Calculus

- Examples: Assume E is a binary relation symbol
  - $(\exists x)E(x,x)$
  - $(\forall x)(\forall y)(\exists z)(E(x,z) \wedge E(z,y))$
  - $(\exists z_1)(\exists z_2)(E(x,z_1) \wedge E(z_1,z_2) \wedge E(z_2,y))$
  - $(\exists y)(\exists z)(E(x,y) \wedge E(x,z) \wedge (y \neq z))$

- Free and bound variables:
  - In the first two formulas above, no variable is free.
  - In the third formula above, the free variables are x and y.
  - In the fourth formula above, the only free variable is x.
  - Intuitively, a variable is free in a formula if the variable must be assigned a value in order to tell if the formula is true or false.

Relational Calculus as a Database Query Language

Definition:

- A relational calculus expression is an expression of the form

$$\{(x_1,\ldots,x_k):\ \varphi(x_1,\ldots x_k)\},$$

  where $\varphi(x_1,\ldots,x_k)$ is a relational calculus formula with $x_1,\ldots,x_k$ as its free variables.
- When applied to a relational database I, this relational calculus expression returns the k-ary relation that consists of all k-tuples $(a_1,\ldots,a_k)$ that make the formula "true" on I.

Example: The relational calculus expression

$$\{(x,y):\ \exists z(E(x,z) \wedge E(z,y))\}$$

returns the set P of all pairs of nodes (a,b) that are connected via a path of length 2.

Relational Calculus as a Database Query Language

Example: FACULTY(name, dpt, salary), CHAIR(dpt, name)
Give a relational calculus expression for C-SALARY(dpt,salary)
(find the salaries of department chairs).

$$\{(x,y):\ \exists u(\text{FACULTY}(u,x,y) \wedge \text{CHAIR}(x,u))\}$$

Here is another relational calculus expression for the same task:

$$\{(x,y):\ \exists u \exists v(\text{FACULTY}(u,x,y) \wedge \text{CHAIR}(x,v) \wedge (u=v))\}$$

Relational Calculus as a Database Query Language

Example: FACULTY(name, dpt, salary)

Find the names of the highest paid faculty in CS

$\{x: \varphi(x)\}$, where $\varphi(x)$ is the formula:

$$\exists y,z\,(\text{FACULTY}(x,y,z) \wedge y = \text{“CS”} \wedge (\forall u,v,w(\text{FACULTY}(u,v,w) \wedge v = \text{“CS”} \rightarrow z \geq w)))$$

Exercise: Express this query in relational algebra and in SQL.

Abbreviation:

- $\exists x_1,\ldots,x_k$ stands for $\exists x_1,\ldots,\exists x_k$
- $\forall x_1,\ldots,x_k$ stands for $\forall x_1,\ldots,\forall x_k$

Natural Join in Relational Calculus

Example: Let R(A,B,C) and S(B,C,D) be two ternary relation schemas.

- Recall that, in relational algebra, the natural join $R \bowtie S$ is given by

$$\pi_{R.A,R.B,R.C,S.D}(\sigma_{R.B = S.B \wedge R.C = S.C}(R \times S)).$$

- Give a relational calculus expression for $R \bowtie S$

$$\{(x_1,x_2,x_3,x_4):\ R(x_1,x_2,x_3) \wedge S(x_2,x_3,x_4)\}$$

Note: The natural join is expressible by a quantifier-free formula of relational calculus.

Quotient in Relational Calculus

- Recall that the quotient (or division) $R \div S$ of two relations R and S is the relation of arity $r - s$ consisting of all tuples $(a_1,\dots,a_{r-s})$ such that for every tuple $(b_1,\dots,b_s)$ in S, we have that $(a_1,\dots,a_{r-s}, b_1,\dots,b_s)$ is in R.

- Assume that R has arity 5 and S has arity 3.

  Express $R \div S$ in relational calculus.

$$\{(x_1,x_2): (\forall x_3)(\forall x_4)(\forall x_5)\ (S(x_3,x_4,x_5) \rightarrow R(x_1,x_2,x_3,x_4,x_5))\}$$

  - Much simpler than the relational algebra expression for $R \div S$

Relational Algebra vs. Relational Calculus

Codd’s Theorem (informal statement):

Relational Algebra and Relational Calculus have essentially the same expressive power, i.e., they can express the same queries.

Note:

- This statement is **not** entirely accurate.
- In what follows, we will give a rigorous formulation of Codd’s Theorem and sketch its proof.

Queries

Definition: Let **S** be a relational database schema.
A k-ary query on **S** is a function q defined on database instances over S such that if I is a database instance over S, then q(I) is a k-ary relation that is invariant under isomorphisms and has values among those occurring in the relations in I
(i.e., if h: I $\rightarrow$ J is an isomorphism, then q(J) = h(q(I)).

Note:

- All "queries" that we have expressed in relational algebra and/or in relational calculus so far are queries in the above formal sense.
- In particular, a relational calculus expression of the form $\{(x_1,\ldots,x_k): \varphi(x_1,\ldots x_k)\}$ defines a k-ary query.

From Relational Algebra to Relational Calculus

Theorem: For every relational expression E, there is an equivalent relational calculus expression $\{(x_1,\ldots,x_k)\colon \varphi(x_1,\ldots x_k)\}$.

Proof: By induction on the construction of rel. algebra expressions.

- If E is a relation R of arity k, then we take $\{(x_1,\ldots,x_k)\colon E(x_1,\ldots,x_k)\}$.
- Assume E_1 and E_2 are expressible by $\{(x_1,\ldots,x_k)\colon \varphi_1(x_1,\ldots,x_k)\}$ and by $\{(x_1,\ldots,x_m)\colon \varphi_2(x_1,\ldots,x_m)\}$. Then
  - $E_1 \cup E_2$ is expressible by $\{(x_1,\ldots,x_k)\colon \varphi_1(x_1,\ldots,x_k) \vee \varphi_2(x_1,\ldots,x_k)\}$.
  - $E_1 - E_2$ is expressible by $\{(x_1,\ldots,x_k)\colon \varphi_1(x_1,\ldots,x_k) \wedge \neg\varphi_2(x_1,\ldots,x_k)\}$.
  - $E_1 \times E_2$ is expressible by $\{(x_1,\ldots,x_k,y_1,\ldots,y_m)\colon \varphi_1\ (x_1,\ldots,x_k) \wedge \varphi_2(y_1,\ldots,y_m)\}$

From Relational Algebra to Relational Calculus

Theorem: For every relational expression E, there is an equivalent relational calculus expression $\{(x_1,\ldots,x_k): \varphi(x_1,\ldots x_k)\}$.

Proof: (continued)

- Assume that E is expressible by $\{(x_1,\ldots,x_k): \varphi(x_1,\ldots,x_k)\}$.
  Then
  - $\pi_{1,3}$(E) is expressible by
    $\{(x_1,x_3): (\exists\, x_2)(\exists\, x_4) \ldots(\exists\, x_k)\ \varphi(x_1,\ldots,x_k)\ \}$
  - σ_{Θ}(E) is expressible by
    $\{(x_1,\ldots,x_k): \Theta^* \wedge \varphi(x_1,\ldots,x_k)\}$, where Θ^* is the rewriting of Θ as a formula of relational calculus.

Corollary: Relational Calculus is relationally complete.

From Relational Calculus to Relational Algebra

Fact: It is **not** true that for every relational calculus expression φ, there is an equivalent relational algebra expression E.

Examples:

- $\{(x_1,\ldots,x_k)\colon \neg R(x_1,\ldots,x_k)\}$
- $\{(x,y)\colon \exists z(\mathrm{CHAIR}(x,z) \wedge y \neq z)\}$, where CHAIR(dpt,name)
- $\{x\colon \forall y,z\ \mathrm{ENROLLS}(x,y,z)\}$, where ENROLLS(s-name,course,term)

From Relational Calculus to Relational Algebra

Note: The previous three relational calculus expression produce different answers when we consider different domains over which the variables are interpreted.

Example: If the variables $x_1,\ldots,x_k$ range over a domain D, then

$$\{(x_1,\ldots,x_k) : \neg R(x_1,\ldots,x_k)\} = D^k - R.$$

Fact:

- The relational calculus expression $\{(x_1,\ldots,x_k) : \neg R(x_1,\ldots,x_k)\}$ is **not** "domain independent".
- The relational calculus expression $\{(x_1,\ldots,x_k) : S(x_1,\ldots,x_k) \wedge \neg R(x_1,\ldots,x_k)\}$ is "domain independent".

From Relational Calculus to Relational Algebra

- Question: How can we go from relational calculus to relational algebra?
- Answer: There are two possibilities:
  - Restrict ourselves to “domain independent” relational calculus expressions.
  - “Relativize” the semantics of relational calculus expressions by fixing a domain over which the variables range.

Active Domain

Definition:

- The active domain adom(φ) of a relational calculus formula φ is the set of all constants that occur in φ.
  - If φ is R(x,y), then adom(φ) = $\emptyset$
  - If φ is $\exists y(R(x,y) \wedge (y > 3) \wedge (x < 5))$, then adom($\varphi$) = {3,5}.

- The active domain adom(I) of a relational database instance I is the set of all values that occur in the relations of I.

Active Domain and Relative Interpretations

Definition: Let $\varphi(x_1,\ldots,x_k)$ be a relational calculus formula and let I be a relational database instance.

- If is D a domain such $adom(\varphi) \cup adom(I) \subseteq D$, then $\varphi^D(I)$ is the result of evaluating $\varphi(x_1,\ldots,x_k)$ over D and I, that is,
  - all variables and quantifiers are assumed to range over D;
  - the relation symbols in φ are interpreted by the relations in I.

- By definition, $\varphi^{adom}(I)$ is $\varphi^D(I)$, where $D = adom(\varphi) \cup adom(I)$.

Active Domain and Relative Interpretation

Example: Let φ be $\neg R(x,y)$ and $I = \{(1,2)\}$.

- $adom(I) = \{1,2\}$
- $\varphi^{adom}(I) = \{(2,1), (1,1), (2,2)\}$
- If $D = \{1,2,3\}$, then

  $\varphi^{D}(I) = \{(2,1),(1,1),(2,2),(3,3),(1,3),(3,1),(2,3),(3,2)\}$

Active Domain and Relative Interpretation

Example: Let φ be $\exists y R(x,y)$ and $I = \{(1,1),(1,2),(2,1),(1,3)\}$.

- $adom(I) = \{1,2,3\}$
- $\varphi^{adom}(I) = \{1,2\}$
- If $D = \{1,2,3,4\}$, then

  $\varphi^{D}(I) = \{1,2\}$.
- More generally, if $adom(I) \subseteq D$, then

  $\varphi^{D}(I) = \{1,2\}$.

Active Domain and Relative Interpretation

Example: Let φ be $\forall y R(x,y)$ and $I = \{(1,1),(1,2),(2,1)\}$.

- $adom(I) = \{1,2\}$
- $\varphi^{adom}(I) = \{1\}$
- If $D = \{1,2,3\}$, then

  $\varphi^{D}(I) = \emptyset$.

Domain Independence

Definition: A relational calculus formula φ is domain independent if for every relational instance I and every domain D such that $adom(\varphi) \cup adom(I) \subseteq D$, we have that

$$\varphi^D(I) = \varphi^{adom}(I).$$

Examples:

- $\neg R(x_1,\ldots,x_k)$ is **not** domain independent.
- $\exists y R(x,y)$ is domain independent.
- $\forall y R(x,y)$ is **not** domain independent.
- $\forall y (R(x,y) \rightarrow y > 5)$ is domain independent.

Equivalence of Relational Algebra and Relational Calculus

Theorem: The following are equivalent for a k-ary query q:

1. There is a relational algebra expression E such that q(I) = E(I), for every database instance I
   (in other words, q is expressible in relational algebra).

2. There is a domain independent relational calculus formula φ such that q(I) = φ^{adom} (I) (in other words, q is expressible in domain independent relational calculus).

3. There is a relational calculus formula ψ such that q(I) = ψ^{adom} (I)
   (in other words, q is expressible in relational calculus under the active domain interpretation).

Equivalence of Relational Algebra and Relational Calculus

Proof (Sketch):

1. $\Rightarrow$ 2. Show by induction that the earlier translation of relational algebra to relational calculus is actually a translation of relational algebra to domain independent relational calculus.

2. $\Rightarrow$ 3. This implication is obvious.

3. $\Rightarrow$ 1.

- Show first that for every relational database schema **S**, there is a relational algebra expression E such that for every database instance I, we have that adom(I) = E(I).

- Use the above fact and induction on the construction of relational calculus formulas to obtain a translation of relational calculus under the active domain interpretation to relational algebra.

Equivalence of Relational Algebra and Relational Calculus

- In this translation, the most interesting part is the simulation of the universal quantifier $\forall$ in relational algebra.
  - It uses the logical equivalence $\forall y \psi \equiv \neg \exists y \neg \psi$
- As an illustration, consider $\forall y R(x,y)$.
  - $\forall y R(x,y) \equiv \neg \exists y \neg R(x,y)$
  - $adom(I) = \pi_1(R) \cup \pi_2(R)$

Rel.Calc. formula φ	Relational Algebra Expression for φ^{adom}
$\neg R(x,y)$	$(\pi_1(R) \cup \pi_2(R)) \times (\pi_1(R) \cup \pi_2(R)) - R$
$\exists y \neg R(x,y)$	$\pi_1((\pi_1(R) \cup \pi_2(R)) \times (\pi_1(R) \cup \pi_2(R)) - R)$
$\neg \exists y \neg R(x,y)$	$(\pi_1(R) \cup \pi_2(R)) - (\pi_1((\pi_1(R) \cup \pi_2(R)) \times (\pi_1(R) \cup \pi_2(R)) - R))$

Equivalence of Relational Algebra and Relational Calculus

Remarks:

- The Equivalence Theorem is effective. Specifically, the proof of this theorem yields two algorithms:
  - an algorithm for translating from relational algebra to domain independent relational calculus, and
  - an algorithm from translating from domain independent relational calculus to relational algebra.

- Each of these two algorithms runs in linear time.

Domain Independent Relational Calculus

Note:

- A desirable feature of a logical formalism is that there is an (efficient) algorithm for determining whether or not an expression is a formula of that formalism.
- Both relational algebra and relational calculus have this property.

Question:

- Does domain independent relational calculus have this property?
- In other words, is there an algorithm such that, given a relational calculus formula φ, the algorithm tells whether or not φ is domain independent?

Domain Independent Relational Calculus

Bad News …

Theorem (Di Paola – 1969): Determining domain independence is an undecidable problem, i.e., there is no algorithm such that, given a relational calculus formula φ, the algorithm tells whether or not φ is domain independent.

Some Good News:

Theorem: Domain independent relational calculus has an effective syntax, i.e., there is a class ***F*** of relational calculus formulas such that:

- There is an (efficient) algorithm for testing membership in ***F***.
- Every formula in ***F*** is domain independent.
- Every domain independent relational calculus formula is logically equivalent to a formula in ***F***.

Domain Independent Relational Calculus

For much more on domain independence:

- Read Sections 5.3 and 5.4 of “Foundations of Databases” by Abiteboul, Hull, and Vianu
- Read the papers
  - “The recursive unsolvability of the decision problem for the class of definite formulas” by Robert A. Di Paola, JACM, Vol. 16, 1969, pages 324-327.
  - “Safety and translation of relational calculus” by Allen van Gelder and Rodney Topor, ACM Transactions on Database Systems, Vol. 16, 1991, pages 235 – 278.

Queries (Revisited)

Definition: Let **S** be a relational database schema.

- A k-ary query on **S** is a function q defined on database instances over S such that if I is a database instance over S, then q(I) is a k-ary relation on adom(I) that is invariant under isomorphisms (i.e., if h: I $\rightarrow$ J is an isomorphism, then q(J) = h(q(I)).

- A Boolean query on **S** is a function q defined on database instances over S such that if I is a database instance over S, then q(I) = 0 or q(I) = 1, and q(I) is invariant under isomorphisms.

Example: The following are Boolean queries on graphs:

- Given a graph E (binary relation), is the diameter of E at most 3?
- Given a graph E (binary relation), is E connected?

Three Fundamental Algorithmic Problems about Queries

- The Query Evaluation Problem: Given a query q and a database instance I, find q(I).

- The Query Equivalence Problem: Given two queries q and q′ of the same arity, is it the case that q $\equiv$ q′ ?

  (i.e., is it the case that, for every database instance I, we have that q(I) = q′(I)?)

- The Query Containment Problem: Given two queries q and q′ of the same arity, is it the case that q $\subseteq$ q′ ?

  (i.e., is it the case that, for every database instance I, we have that q(I) $\subseteq$ q′(I)?)

Three Fundamental Algorithmic Problems about Queries

- The Query Evaluation Problem is the main problem in query processing.
- The Query Equivalence Problem underlies query processing and optimization, as we often need to transform a given query to an equivalent one.
- The Query Containment Problem and Query Equivalence Problem are closely related to each other:
  - $q \equiv q'$ if and only if $q \subseteq q'$ and $q' \subseteq q$.
  - $q \subseteq q'$ if and only if $q \equiv q \wedge q'$.

Three Fundamental Algorithmic Problems about Queries

- Our goal is to investigate the algorithmic aspects of these problems for queries expressible in relational algebra/relational calculus.
- The questions we want to address are:
  - How can we measure the precise "difficulty" of these problems?
  - Are there "good" algorithms for solving these problems?
  - If not, are there special cases of these problems for which "good" algorithms exist?

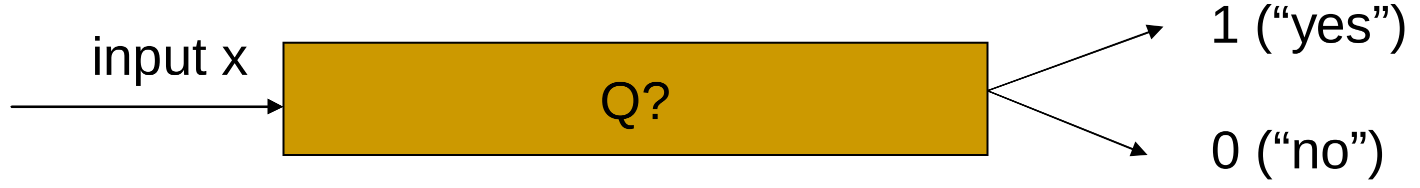
Three Fundamental Algorithmic Problems about Queries

Our study of these problems will use concepts and methods from two different, yet related, areas:

- Mathematical Logic:
  - Computability Theory and Undecidable Problems
- Computational Complexity Theory:
  - Complexity Classes and Complete Problems
  - In particular, the classes P and NP, and NP-complete problems.

Decision Problems and Languages

- Definition (informal): A decision problem Q consists of a set of inputs and a question with a “yes” or “no” answer for each input.



- Definition:
  - Σ^* is the set of all strings over a finite alphabet Σ.
  - A language over Σ is a set $L \subseteq \Sigma^*$
  - Every language L gives rise to the following decision problem:
    - Given $x \in \Sigma^*$, is $x \in L$?
  - Conversely, every decision problem can be thought of as arising from a language, namely,
    the language consisting of all inputs with a “yes” answer.

Turing Computability

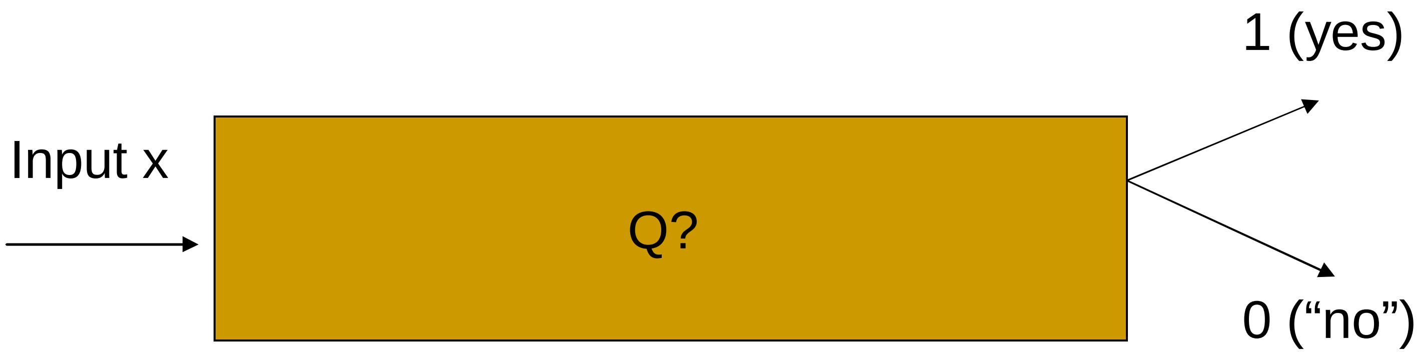
- Turing machines
- Turing computable (partial) functions f: $\Sigma^* \to \Sigma^*$
- Church's Thesis (aka Church-Turing Thesis): The following statements are equivalent for a (partial) function f: $\Sigma^* \to \Sigma^*$:
  - There is a Turing machine that computes f
  - There is an algorithm that computes f.
- Main Use of Church's Thesis: To show that there is no algorithm for computing a function f, it suffices to show that there is no Turing machine that computes f.

Recursive and Recursively Enumerable Languages

- Definition: Let $L \subseteq \Sigma^*$ be a language
  - L is recursive if its characteristic function χ is Turing computable, where
    - $\chi_L(x) = 1$ if $x \in L$
    - $\chi_L(x) = 0$ if $x \notin L$.
  - L is recursively enumerable if its semi-characteristic function s_L is Turing computable, where
    - $s_L(x) = 1$ if $x \in L$
    - $s_L(x) =$ undefined if $x \notin L$.
- Theorem: The following are equivalent for a language $L \subseteq \Sigma^*$:
  - L is recursive.
  - Both L and its complement $\Sigma^* - L$ are recursively enumerable.

Decidable and Undecidable Problems

- Definition: Let Q be a decision problem.
  - Q is decidable (solvable) if the language associated with Q is recursive.
  - Q is undecidable (unsolvable) if the language associated with Q is not recursive.



Q is undecidable means that there is **no** algorithm for this problem

Undecidable Problems

Fact: Undecidable problems exist.

Proof: Use a counting argument:

- There are countably many Turing machines.
- There are uncountably many languages $L \subseteq \{0,1\}^*$.

Theorem: Many natural problems of algorithmic interest or of mathematical significance are undecidable.

Undecidable Problems

Theorem: The following problems are undecidable:

- The Halting Problem (A. Turing – 1936): Given a Turing machine M and an input x, does M halt on x?
- The Finite Validity Problem (B. Trakhtenbrot – 1949): Given a first-order formula φ on graphs, is φ true on every finite graph?
- Hilbert's 10th Problem (Y. Matijacevic – 1971): Given a multivariate polynomial $p(x_1,\ldots,x_n)$ with integer coefficients, does $p(x_1,\ldots,x_n)$ have an all-integers solution?

Undecidable Problems

- The Halting Problem (A. Turing – 1936): Given a Turing machine M and an input x, does M halt on x?

- Implications of Undecidability of the Halting Problem:
  - The undecidability of the Halting Problem implies that there is **no** algorithm such that, given a C program p and an input x, the algorithm determines whether the program p produces an output on input x or goes into an infinite loop.
  - Of course, it may still be possible to show that a particular program terminates on a given input (or even on every input), but it is **not** possible to automate this process for every program.
  - But even this may be a difficult task …

Proving Program Termination

- McCarthy's Program

  Given a positive integer n:

  While n > 1, do:

  - If n is even, then set n: = n/2;
  - If n is odd, then set n:= 3n+1.

- Example Run:
  - n = 11 $\mapsto$ 34 $\mapsto$ 17 $\mapsto$ 52 $\mapsto$ 26 $\mapsto$ 13 $\mapsto$ 40 $\mapsto$ 20 $\mapsto$ 10 $\mapsto$ 5 $\mapsto$ 16 $\mapsto$ 8 $\mapsto$ 4 $\mapsto$ 2 $\mapsto$ 1.

- Open Problem: Does this program terminate on every input?

Undecidable Problems

- The Finite Validity Problem (B. Trakhtenbrot – 1949): Given a first-order formula φ on graphs, is φ true on every finite graph?
- Examples of Finitely Valid Formulas:
  - $\forall x(E(x,x) \rightarrow \exists yE(x,y))$
  - $\forall x\forall y(E(x,x) \wedge x = y \rightarrow E(y,y))$
  - "if E is a total order, then E has a biggest element"
- Example of Non-Finitely Valid Formulas:
  - $\forall x \forall y (E(x,y) \rightarrow E(y,x))$
  - $(\forall x \exists y E(x,y)) \rightarrow (\exists y\forall x E(x,y))$
- The undecidability of the Finite Validity Problem implies that there is **no** algorithm for telling formulas in the first group from formulas in the second group.

Undecidable Problems

- Hilbert's 10th Problem (Y. Matijacevic – 1971): Given a multivariate polynomial $p(x_1,\dots,x_n)$ with integer coefficients, does $p(x_1,\dots,x_n)$ have an all-integers root ? (i.e., does the equation $p(x_1,\dots,x_n) = 0$ have an all integer solution?)
- Diophantine Equations (Diophantus of Alexandria 3rd Century AD)
  - $3x + 5y - 8z = 0$
  - $x^2 - 2xy + z^3 + 9 = 0$
  - $x^2 - 100y^2 + 1 = 0$
  - $x^2 + y^2 - z^2 = 0$
  - $x^3 + y^3 - z^3 = 0$
- The undecidability of Hilbert's 10th Problem implies that there is **no** algorithm to tell whether or not a given Diophantine equation has a solution consisting entirely of integers.

Undecidable Problems

Note:

- The Halting Problem is recursively enumerable, but not recursive

  (hence, its complement is not recursively enumerable).

- The Finite Validity Problem is co-recursively enumerable, but not recursive.

  (hence, it is not even recursively enumerable).

- Hilbert's 10th Problem is recursively enumerable, but not recursive

  (hence, its complement is not recursively enumerable).

The Reduction Method

- By now there is a vast library of undecidable problems.
- The Reduction Method is the main technique for establishing undecidability.
- Reduction Method: To show that a language L* is not recursive, it suffices to find a non-recursive language L and a total Turing computable function f such that for every string x, we have that

$$x \in L \quad \Leftrightarrow \quad f(x) \in L^*.$$

  - Such a function f is called a reduction of L to L*
  - $L \preccurlyeq L^*$ means that there is a reduction of L to L*.

The Reduction Method

- The Halting Problem was the first fundamental decision problem shown to be undecidable.
- The Finite Validity Problem was shown to be undecidable by showing that Halting Problem $\preccurlyeq$ Finite Validity Problem.
- Many database problems have been shown to be undecidable via reductions from one the following problems:
  - The Halting Problem
  - The Finite Validity Problem
  - Hilbert's 10th Problem.
- In particular, Di Paola proved that the Domain Independence Problem for relational calculus formulas is undecidable by showing that Finite Validity Problem $\preccurlyeq$ Domain Independence.

Undecidability of The Query Equivalence Problem

- The Query Equivalence Problem: Given two queries q and q′ of the same arity, is it the case that $q \equiv q'$?

  (i.e., is q(I) = q′(I) on every database instance I?)

- Theorem: The Query Equivalence Problem for relational calculus queries is undecidable.

  Proof: Finite Validity Problem $\preccurlyeq$ Query Equivalence Problem
  - To see, this let ψ^* be a fixed finitely valid relational calculus sentence (say, $\forall x(E(x,x) \rightarrow \exists yE(x,y))$).
  - Then, for every relational calculus sentence φ, we have that

    $$\varphi \text{ is finitely valid} \Leftrightarrow \varphi \equiv \psi^*.$$

Undecidability of the Query Containment Problem

- The Query Containment Problem: Given two queries q and q′ of the same arity, is it the case that $q \subseteq q'$?

  (i.e., is $q(I) \subseteq q'(I)$ on every database instance I?)

- Corollary: The Query Containment Problem for relational calculus queries in undecidable.

  Proof: Query Equivalence $\preccurlyeq$ Query Containment, since

  $$q \equiv q' \Leftrightarrow q \subseteq q' \text{ and } q' \subseteq q.$$

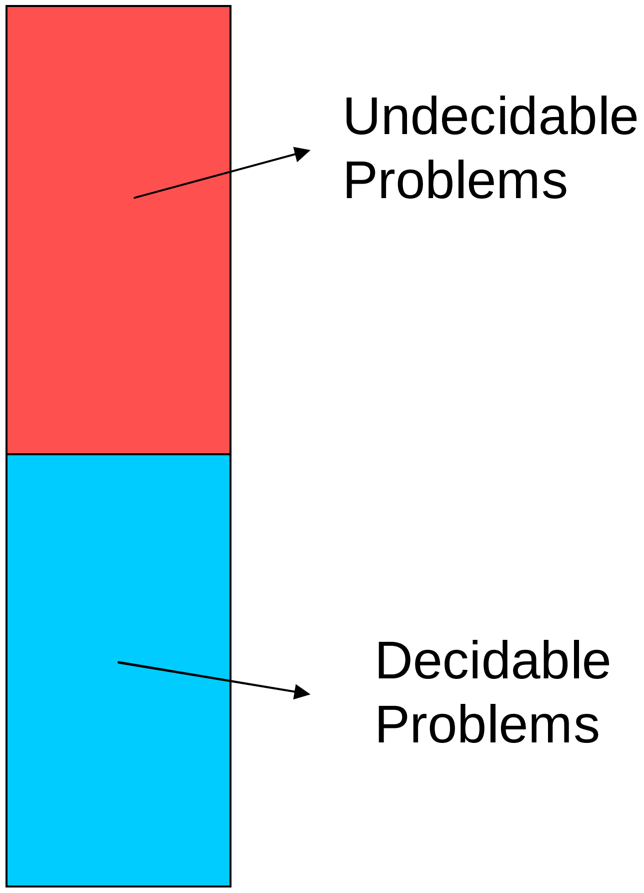
- Notice the chain of reductions:

  Halting Problem $\preccurlyeq$ Finite Validity $\preccurlyeq$ Query Equiv. $\preccurlyeq$ Query Cont.

The Query Evaluation Problem

- The Query Evaluation Problem: Given a query q and a database instance I, find q(I).
- The Query Evaluation Problem for relational calculus queries is decidable, but, as we will see, it has high computational complexity.
- To understand the precise algorithmic difficulty of the Query Evaluation Problem, we need some basic notions and results from computational complexity.

Decidable Problems and Computational Complexity



- Computational Complexity is the quantitative study of decidable problems.
- "From these and other considerations grew our deep conviction that **there must be quantitative laws that govern the behavior of information and computing**. The results of this research effort were summarized in our first paper on this topic, which also named this new research area, "On the computational complexity of algorithms"."

J. Hartmanis, Turing Award Lecture, 1993

Computational Complexity Classes

- Decidable problems are grouped together in computational complexity classes.
- Each computational complexity class consists of all problems that can be solved in a computational model under certain restrictions on the resources used to solve the problem.
- Examples of computational models:
  - Turing Machine TM (deterministic Turing machine)
  - Non-deterministic Turing machine NTM
  - …
- Examples of resources:
  - Amount of time needed to solve the problem
  - Amount of space (memory) needed to solve the problem.
  - …

The Five Basic Computational Complexity Classes

- LOGSPACE (or, L): All decision problems solvable by a TM using extra memory bounded by a logarithmic amount in the input size.
- NLOGSPACE (or, NL): All decision problems solvable by a NTM using extra memory bounded by a logarithmic amount in the input size.
- P (or, PTIME): All decision problems solvable by a TM in time bounded by some polynomial in the input size.
- NP: All decision problems solvable by a NTM in time bounded by some polynomial in the input size.
- PSPACE: All decision problems solvable by a TM using memory bounded by a polynomial in the input size.

The Five Basic Computational Complexity Classes

Theorem:

- The following inclusions hold:
  LOGSPACE $\subseteq$ NLOGSPACE $\subseteq$ P $\subseteq$ NP $\subseteq$ PSPACE.

- Moreover, it is known that LOGSPACE $\subset$ PSPACE.

- **No** other proper inclusion between these classes is known at present. In particular, it is **not** known whether P = NP.

Note:

- The question: “is P = NP?” is the central open problem in computational complexity.
- It is one of the Millennium Prize Problems – see http://www.claymath.org/millennium/

Complete Problems

- A key property of most complexity classes is that they possess complete problems.

- Intuitively, complete problems are the "hardest" problems in the class in the sense that every other problem can be reduced to it.

- Definition: Let ***C*** be a complexity class.
  A decision problem Q is ***C***-complete if
  - Q is in ***C***.
  - If Q′ is in ***C***, then there is a "suitable" total Turing computable function f such that for every string x, we have that

    $$x \in Q' \iff f(x) \in Q.$$

    - "suitable" means that f can be computed with fewer resources than those used to define ***C***.
    - So, f is a reduction of a restricted nature.

Complete Problems and Reductions

Complexity Class	Reductions for Complete Problems
PSPACE	Polynomial-time computable
NP	Polynomial-time computable
P	Logspace-computable
NL	Logspace-computable

- Definition: A decision problem Q is NP-complete if
  - Q is in NP.
  - If Q′ is in NP, then there is a polynomial-time computable function f such that for every string x, we have that

$$x \in Q' \iff f(x) \in Q.$$

- Such an f is a polynomial-time reduction of Q′ to Q ($Q' \preccurlyeq_p Q$).

Complete Problems for Computational Complexity Classes

- PSPACE-complete:
  - Quantified Boolean Formulas (QBF): Given a quantified Boolean formula $\forall x_1 \exists x_2 \ldots. \forall x_k \varphi$, is it true?
- NP-complete:
  - Satisfiability (SAT): Given a CNF formula φ, is it satisfiable?
  - 3-Colorability: Given a graph G=(V,E), is it 3-colorable?
  - Integer Linear Inequalities (ILI): Given a system of linear inequalities with integer coeffs., does it have an integer solution?
- P-complete:
  - Horn SAT: Given a Horn CNF formula φ, is it satisfiable?
  - Linear Inequalities (LI): Given a system of linear inequalities with integer coefficients, does it have a rational solution?
- NL-complete:
  - Directed Graph Reachability: Given a directed graph G=(V,E) and two nodes s and t, is there a path from s to t?

Polynomial-Time Reductions

- 3-Satisfiability (3SAT): Given a 3CNF formula φ, is it satisfiable? (each clause has at most 3 literals)
- Theorem: 3SAT is NP-complete

  Proof: Show that SAT $\preccurlyeq_p$ 3SAT
  - Let φ be a CNF formula $c_1 \wedge c_2 \dots \wedge c_m$
  - If a clause c_i has more than three literals, then we replace it with a set of clauses each with three literals and certain new variables.
  - For example, if c_i is $(x_1 \vee \neg x_2 \vee x_3 \vee x_4 \vee x_5)$, then we replace c_i by $(x_1 \vee \neg x_2 \vee y_1)$, $(\neg y_1 \vee x_3 \vee y_2)$, $(\neg y_2 \vee x_4 \vee x_5)$.
  - Let φ^* be the resulting 3CNF formula. Then

    φ is satisfiable $\Leftrightarrow$ φ^* is satisfiable (check this).

Complete Problems for Computational Complexity Classes

- Proposition: Let $\mathcal{C}$ and $\mathcal{C}'$ be one of the complexity classes NLOGSPACE, P, NP, PSPACE such that $\mathcal{C} \subseteq \mathcal{C}'$ and let Q be a $\mathcal{C}'$-complete problem. Then the following statements are equivalent:
  - $\mathcal{C} = \mathcal{C}'$.
  - Q is in $\mathcal{C}$.

  In particular, the following statements are equivalent:
  - P = NP.
  - SAT is in P
  - Your favorite NP-complete problem is in P.

- Conclusion:
  - Complete problems hold the secret of whether or not a higher computational complexity class collapses to a lower one.
  - Showing that a decision problem Q is NP-complete provides **strong evidence** that Q is **not** in P.

Complexity of the Query Evaluation Problem

- The Query Evaluation Problem for Relational Calculus:
  Given a relational calculus formula φ and a database instance I, find $\varphi^{adom}(I)$.

- The Query Evaluation Problem for Relational Algebra:
  Given a relational algebra expression E and a database instance I, find E(I).

- Theorem: The Query Evaluation Problem for Relational Calculus is PSPACE-complete.

- Corollary: The Query Evaluation Problem for Relational Algebra is PSPACE-complete.

Complexity of the Query Evaluation Problem

- Theorem: The Query Evaluation Problem for Relational Calculus is PSPACE-complete.

  Proof: We need to show that

  - This problem is in PSPACE (i.e., give a PSPACE-algorithm for it).
  - This problem is PSPACE-hard.

  We start with the second task.

Complexity of the Query Evaluation Problem

- Theorem: The Query Evaluation Problem for Relational Calculus is PSPACE-hard.
- Proof: Show that

  Quantified Boolean Formulas $\preccurlyeq_p$ Query Evaluation for Rel. Calc.

  Given QBF $\forall x_1 \exists x_2 \ldots . \forall x_k \psi$
  - Let V and P be two unary relation symbols
  - Obtain ψ^* from ψ by replacing x_i by $P(x_i)$, and $\neg x_i$ by $\neg P(x_i)$
  - Let I be the database instance with V = {0,1}, P={1}.
  - Then the following statements are equivalent:
    - $\forall x_1 \exists x_2 \ldots . \forall x_k \psi$ is true
    - $\forall x_1 (V(x_1) \rightarrow \exists x_2 (V(x_2) \wedge (\ldots \forall x_k (V(x_k) \rightarrow \psi^*))\ldots)$ is true on I.

Complexity of the Query Evaluation Problem

- Theorem: The Query Evaluation Problem for Relational Calculus is in PSPACE.

  Proof (Hint): Let φ be a relational calculus formula $\forall x_1 \exists x_2 \ldots \forall x_m \psi$ and let I be a database instance.
  - Exponential Time Algorithm: We can find $\varphi^{adom}(I)$, by exhaustively cycling over all possible interpretations of the x_i's.

    This runs in time $O(n^m)$, where $n = |I|$ (size of I).
  - A more careful analysis shows that this algorithm can be implemented in $O(m \cdot \log n)$-space.
    - Use m blocks of memory, each holding one of the n elements of adom(I) written in binary (so O(logn) space is used in each block).
    - Maintain also m counters in binary to keep track of the number of elements examined.

$\forall x_1$	$\exists x_2$	…	$\forall x_m$
a_1 in adom(I) written in binary	a_2 in adom(I) written in binary	…	a_m in adom(I) written in binary

Complexity of the Query Evaluation Problem

- Corollary: The Query Evaluation Problem for Relational Algebra is PSPACE-complete.

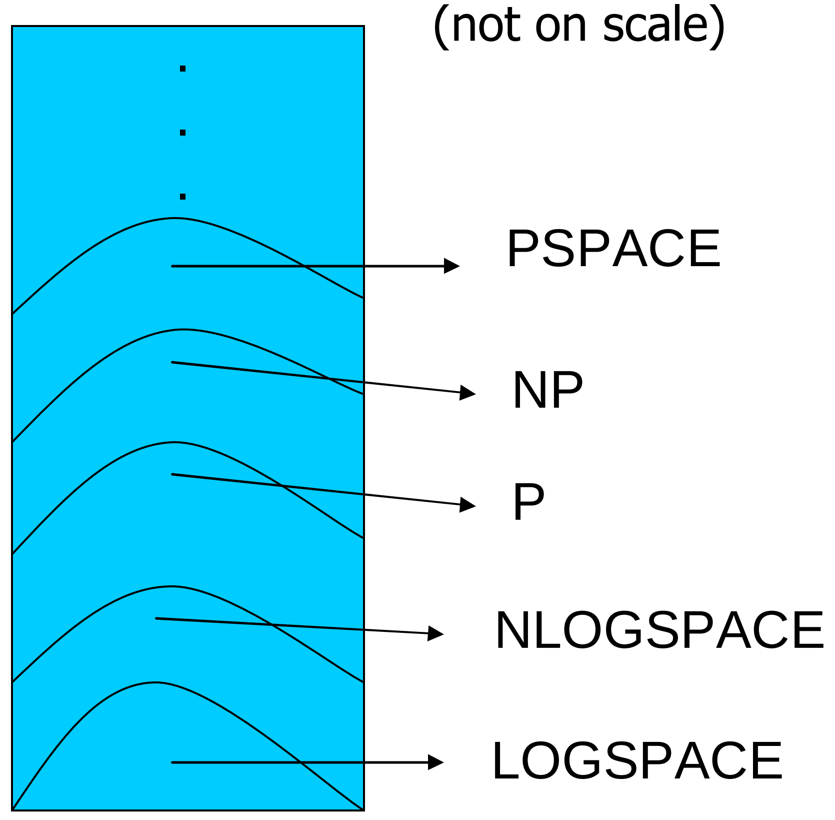
  Proof: The translation of relational calculus to relational algebra yields a polynomial-time reduction of the Query Evaluation Problem for Relational Calculus to the Query Evaluation Problem for Relational Algebra.

Summary

- The Query Evaluation Problem for Relational Calculus is PSPACE-complete.
- The Query Equivalence Problem for Relational Calculus in undecidable.
- The Query Containment Problem for Relational Calculus is undecidable.

Computational Complexity Classes

Classification of Decidable Problems (not on scale)



There are many other complexity classes. For a comprehensive catalog, visit the Complexity Zoo at

qwiki.stanford.edu/wiki/Complexity_Zoo

Complete Problems

- A key property of most complexity classes is that they possess complete problems.

- Intuitively, complete problems are the "hardest" problems in the class in the sense that every other problem can be reduced to it.

- Definition: Let ***C*** be a complexity class.

  A decision problem Q is ***C***-complete if
  - Q is in ***C***.
  - If Q′ is in ***C***, then there is a "suitable" total Turing computable function f such that for every string x, we have that

$$x \in Q' \iff f(x) \in Q.$$

    - "suitable" means that f can be computed with fewer resources than those used to define ***C***.
    - So, f is a reduction of a restricted nature.

Complete Problems and Reductions

Complexity Class	**Reductions for Complete Problems**
PSPACE	Polynomial-time computable
NP	Polynomial-time computable
P	Logspace-computable
NL	Logspace-computable

- Definition: A decision problem Q is NP-complete if
  - Q is in NP.
  - If Q' is in NP, then there is a polynomial-time computable function f such that for every string x, we have that

$$x \in Q' \iff f(x) \in Q.$$

- Such an f is a polynomial-time reduction of L to L* ($L \preccurlyeq_p L^*$)

The Query Evaluation Problem Revisited

- Since the Query Evaluation Problem for Relational Calculus is PSPACE-hard, there are **no** polynomial-time algorithms for this problem, unless PSPACE = P (which is considered highly unlikely).
- Let's take another look at the exponential-time algorithm for this problem:
  - Let φ be a relational calculus formula $\forall x_1 \exists x_2 \ldots \forall x_m \psi$ and let I be a database instance.
  - Exponential Time Algorithm: We can find $\varphi^{adom}(I)$, by exhaustively cycling over all possible interpretations of the x_i's. This runs in time $O(n^m)$, where $n = |I|$).
  - So, the running time is $O(|I|^{|\varphi|})$, where $|I|$ is the size of I and $|\varphi|$ is the size of the relational calculus formula φ.
  - This tells that the source of exponentiality is the formula size.

The Query Evaluation Problem Revisited

- Theorem: Let φ be a fixed relational calculus formula. Then the following problem is solvable in polynomial time: given a database instance I, find $\varphi^{adom}(I)$. In fact, this problem is in LOGSPACE.
- Proof: Let φ be a fixed relational calculus formula $\forall x_1 \exists x_2 \ldots \forall x_m \psi$
  - The previous algorithm has running time $O(|I|^{|\varphi|})$, which is a polynomial, since now $|\varphi|$ is a constant.
  - Moreover, the algorithm can now be implemented using logarithmic-space only, since we need only maintain a constant number of memory blocks, each of logarithmic size

$\forall x_1$	$\exists x_2$	…	$\forall x_m$
a_1 in adom(I) written in binary	a_2 in adom(I) written in binary	…	a_m in adom(I) written in binary

Vardi’s Taxonomy of the Query Evaluation Problem

M.Y Vardi, “The Complexity of Relational Query Languages”, 1982

- Definition: Let L be a database query language.
  - The combined complexity of L is the decision problem:
    given an L-sentence and a database instance I, is φ true on I?
    (does I satisfy φ?) (in symbols, does $I \models \varphi$?)
  - The data complexity of L is the family of the following decision problems Q_φ, where φ is an L-sentence:
    given a database instance I, does $I \models \varphi$?
  - The query complexity of L is the family of the following decision problems Q_I, where I is a database instance:
    given an L-sentence φ, does $I \models \varphi$?

Vardi's Taxonomy of the Query Evaluation Problem

Note: Let L be a database query language

- The input to the combined complexity problem consists of two parts: an L-sentence and a database instance.

- The input to a member of the data complexity of L consists of a database instance only (the L-sentence is fixed).
  - Hence, the data complexity of L is a special case of the combined complexity of L.

- The input to a member of the query complexity of L consists of an L-sentence only (the database instance is fixed).
  - Hence, the query complexity of L is a special case of the combined complexity of L.

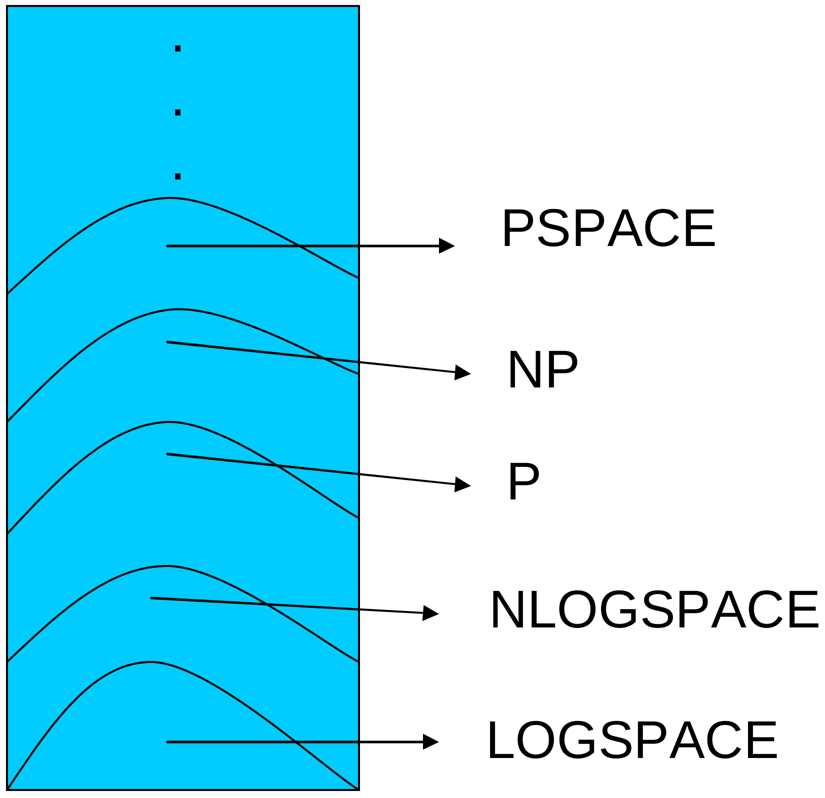
Vardi’s Taxonomy of the Query Evaluation Problem

- Definition: Let L be a database query language and let C be a computational complexity class.
  - The data complexity of L is in C if for each L-sentence φ, the decision problem Q_φ is in C.
  - The query complexity of L is in C if for every database instance, the decision problem Q_I is in C.
- Vardi’s discovery:

  For most query languages L:
  - The data complexity of L is of lower complexity than both the combined complexity of L and the query complexity of L
  - The query complexity of L can be as hard as the combined complexity of L.

Taxonomy of the Query Evaluation Problem for Relational Calculus

Computational Complexity Classes



The Query Evaluation Problem for Relational Calculus

Problem	**Complexity**
Combined Complexity	PSPACE-complete
Query Complexity	■ Is in PSPACE ■ It can be PSPACE-complete
Data Complexity	In LOGSPACE

The Query Evaluation Problem for Relational Calculus

- **Paradox:**
  - The Query Evaluation Problem for Relational Calculus has very high combined complexity (PSPACE-complete, so "harder" than NP-complete).
  - Yet, database systems evaluate SQL queries "efficiently".
- **Resolution of the Paradox:**
  - In practice, we deal with the data complexity of the Query Evaluation Problem for Relational Calculus, because we typically have a small fixed collection of queries to answer (while of course the database instances vary).
  - The data complexity of the Query Evaluation Problem for Relational Calculus is in LOGSPACE (hence, in PTIME); so, in principle, it is a tractable problem.

Sublanguages of Relational Calculus

- Question: Are there interesting sublanguages of relational calculus for which the Query Containment Problem and the Query Evaluation Problem are "easier" than the full relational calculus?

- Answer:
  - Yes, the language of conjunctive queries is such a sublanguage.
  - Moreover, conjunctive queries are the most frequently asked queries against relational databases.

Conjunctive Queries

- **Definition:** A conjunctive query is a query expressible by a relational calculus formula in prenex normal form built from atomic formulas $R(y_1,\ldots,y_n)$, and $\wedge$ and $\exists$ only.

  $$\{(x_1,\ldots,x_k)\colon\ \exists z_1 \ldots \exists z_m\ \chi(x_{\mathbf{1}}, \ldots,x_k, z_1,\ldots,z_k)\},$$

  where $\chi(x_{\mathbf{1}}, \ldots,x_k, z_1,\ldots,z_k)$ is a conjunction of atomic formulas of the form $R(y_1,\ldots,y_m)$.
  - Equivalently, a conjunctive query is a query expressible by a relational algebra expression of the form

    $\pi_X(\sigma_\Theta(R_1\times \ldots\times R_n))$, where

    Θ is a conjunction of equality atomic formulas (equijoin).
  - Equivalently, a conjunctive query is a query expressible by an SQL expression of the form

    SELECT <list of attributes>

    FROM <list of relation names>

    WHERE <conjunction of equalities>

Conjunctive Queries

- **Definition:** A conjunctive query is a query expressible by a relational calculus formula in prenex normal form built from atomic formulas $R(y_1,\ldots,y_n)$, and $\wedge$ and $\exists$ only.

  $$\{(x_1,\ldots,x_k)\colon\ \exists z_1 \ldots \exists z_m\ \chi(x_{\mathbf{1}}, \ldots, x_k, z_1,\ldots,z_k)\}$$

  - A conjunctive query can be written as a logic-programming rule:

    $Q(x_1,\ldots,x_k)$:-- $R_1(\mathbf{u}_1), \ldots, R_n(\mathbf{u}_n)$, where

    - Each variable x_i occurs in the right-hand side of the rule.
    - Each $\mathbf{u}_i$ is a tuple of variables (not necessarily distinct)
    - The variables occurring in the right-hand side (the body), but not in the left-hand side (the head) of the rule are existentially quantified (but the quantifiers are not displayed).
    - "," stands for conjunction.

Conjunctive Queries

Examples:

- Path of Length 2: (Binary query)
  $\{(x,y): \exists z (E(x,z) \wedge E(z,y))\}$
  - As a relational algebra expression,
    $\pi_{1,4}(\sigma_{\$2 = \$3} (E \times E))$
  - As a rule:
    q(x,y) :-- E(x,z), E(z,y)
- Cycle of Length 3: (Boolean query)
  $\exists x \exists y \exists z(E(x,y) \wedge E(y,z) \wedge E(z,x))$
  - As a rule (the head has no variables)
    - Q :-- E(x,z), E(z,y), E(z,x)

Conjunctive Queries

- Every relational join is a conjunctive query:
  P(A,B,C), R(B,C,D) two relation symbols
  - $P \bowtie R = \{(x,y,z,w):\ P(x,y,z) \wedge R(y,z,w)\}$
  - q(x,y,z,w) :-- P(x,y,z), R(y,z,w)
    (no variables are existentially quantified)
  - SELECT P.A, P.B, P.C, R.D
    FROM P, R
    WHERE P.B = R.B AND P.C = R.C
- Conjunctive queries are also known as SPJ-queries
  (SELECT-PROJECT-JOIN queries)

Conjunctive Query Evaluation and Containment

- Definition: Two fundamental problems about CQs
  - Conjunctive Query Evaluation (CQE):
    Given a conjunctive query q and an instance I, find q(I).

  - Conjunctive Query Containment (CQC):
    - Given two k-ary conjunctive queries q_1 and q_2, is it true that $q_1 \subseteq q_2$? (i.e., for every instance I, we have that $q_1(I) \subseteq q_2(I)$)
    - Given two Boolean conjunctive queries q_1 and q_2, is it true that $q_1 \models q_2$? (that is, for all I, if $I \models q_1$, then $I \models q_2$)?
      CQC is logical implication.

CQE vs. CQC

- Recall that for relational calculus queries:
  - The Query Evaluation Problem is PSPACE-complete (combined complexity).
  - The Query Containment Problem is undecidable.
- Theorem**:** Chandra & Merlin, 1977
  - CQE and CQC are the "same" problem.
  - Moreover, each is an NP-complete problem.
- Question: What is the common link?
- Answer: The Homomorphism Problem

Isomorphisms Between Database Instances

- Definition: Let I and J be two database instances over the same relational schema **S**.
  - An isomorphism h: I $\rightarrow$ J is a function h: adom(I) $\rightarrow$ adom(J) such that
    - h is one-to-one and onto.
    - For every relational symbol P of S and every $(a_1,\ldots,a_m)$, we have that

      $(a_1,\ldots,a_m) \in P^I$ if and only if $(h(a_1), .., h(a_m)) \in P^J$.
  - I and J are isomorphic if an isomorphism h from I to J exists.
- Note: Intuitively, two database instances are isomorphic if one can be obtained from the other by renaming the elements of its active domain in a 1-1 way.

A Digression to The Isomorphism Problem

- The Isomorphism Problem: Given two database instances I and J over the same relational schema, is I isomorphic to J?

- Fact: The exact computational complexity of the isomorphism problem is **not** known at present.
  - The isomorphism problem is in NP (this is easy).
  - The isomorphism problem is **not** known to be NP-complete (and there is evidence that it is not NP-complete).
  - The isomorphism problem is **not** known to be in P.

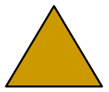
    Finding a polynomial-time algorithm for the isomorphism problem would be a major breakthrough.
  - Special cases of the isomorphism problem are known to be in P (for example, the isomorphism problem on planar graphs).

Homomorphisms

- Definition: Let I and J be two database instances over the same relational schema S.
  A homomorphism h: I $\rightarrow$ J is a function h: adom(I) $\rightarrow$ adom(J) such That for every relational symbol P of S and every $(a_1,\ldots,a_m)$, we have that
  if $(a_1,\ldots,a_m) \in P^I$, then $(h(a_1), .., h(a_m)) \in P^J$.

- Note: The concept of homomorphism is a relaxation of the concept of isomorphism, since every isomorphism is also a homomorphism, but not vice versa.

- Example:
  - A graph G = (V,E) is 3-colorable
    if and only if
    there is a homomorphism h: G $\rightarrow$ K_3



Homomorphisms

- Fact: Homomorphisms compose, i.e.,
  if f: I $\rightarrow$ J and g: J $\rightarrow$ K are homomorphisms, then
  g$\circ$f: I $\rightarrow$ K is a homomorphims, where g$\circ$f(a) = g(f(a)).

- Definition:
  - Two database instances I and I' are homomorphically equivalent if there is a homomorphism h: I $\rightarrow$ I' and a homomorphism h': I' $\rightarrow$ I.
  - I $\equiv_h$ I' means that I and I' are homomorphically equivalent.

- Note: I $\equiv_h$ I' does **not** imply that I and I' are isomorphic.

Homomorphisms

- Fact: Homomorphisms compose, i.e.,
  if f: I $\rightarrow$ J and g: J $\rightarrow$ K are homomorphisms, then
  g$\circ$f: I $\rightarrow$ K is a homomorphims, where g$\circ$f(a) = g(f(a)).

- Definition:
  - Two database instances I and I′ are homomorphically equivalent if there is a homomorphism h: I $\rightarrow$ I′ and a homomorphism h′: I′ $\rightarrow$ I.
  - I $\equiv_h$ I′ means that I and I′ are homomorphically equivalent.

- Note: I $\equiv_h$ I′ does **not** imply that I and I′ are isomorphic.

The Homomorphism Problem

- Definition: The Homomorphism Problem
  Given two database instances I and J, is there a homomorphism
  h: I $\rightarrow$ J?

- Notation: I $\rightarrow$ J denotes that a homomorphism from I to J exists.

- Theorem: The Homomorphism Problem is NP-complete
  Proof: Easy reduction from 3-Colorabilty
  G is 3-colorable if and only if G $\rightarrow$ K_3.

- Exercise: Formulate 3SAT as a special case of the Homomorphism Problem.

The Homomorphism Problem

- Note: The Homomorphism Problem is a fundamental algorithmic problem:
  - Satisfiability can be viewed as a special case of it.
  - k-Colorability can be viewed as a special case of it.
  - Many AI problems, such as planning, can be viewed as a special case of it.
  - In fact, every constraint satisfaction problem can be viewed as a special case of the Homomorphism Problem
    (Feder and Vardi – 1993).

The Homomorphism Problem and Conjunctive Queries

- Theorem**:** Chandra & Merlin, 1977
  - CQE and CQC are the "same" problem.
  - Moreover, each is an NP-complete problem.

- Question: What is the common link?
- Answer:
  - Both CQE and CQC are "equivalent" to the Homomorphism Problem.
  - The link is established by bringing into the picture
    - Canonical conjunctive queries and
    - Canonical database instances.

Canonical CQs and Canonical Instances

- Definition: Canonical Conjunctive Query
  Given an instance I = (R_1, …,R_m), the canonical CQ of I is the Boolean conjunctive query Q^I with (a renaming of) the elements of I as variables and the facts of I as conjuncts, where a fact of I is an expression
  $R_i(a_1,\dots,a_m)$ such that $(a_1,\dots,a_m) \in R_i$.

- Example:
  I consists of E(a,b), E(b,c), E(c,a)

  - Q^I is given by the rule:
    Q^I :-- E(x,z), E(z,y), E(y,x)
  - Alternatively, Q^I is
    $$\exists x \exists y \exists z (E(x,z) \wedge E(z,y) \wedge E(y,x))$$

Canonical Conjunctive Query

- Example: K_3, the complete graph with 3 nodes

  K_3 is a database instance with one binary relation E, where

  $$E = \{(b,r), (r,b), (b,g), (g,b), (r,g), (g,r)\}$$

- The canonical conjunctive query Q^{K3} of K_3 is given by the rule:

  $$Q^{K3} \text{ :- } E(x,y),E(y,x),E(x,z),E(z,x),E(y,z),E(z,y)$$

- The canonical conjunctive query Q^{K3} of K_3 is also given by the relational calculus expression:

  $$\exists x,y,z(E(x,y) \wedge E(y,x) \wedge E(x,z) \wedge E(z,x) \wedge E(y,z) \wedge E(z,y))$$

Canonical Database Instance

- Definition: Canonical Instance

  Given a CQ Q, the canonical instance of Q is the instance I^Q with the variables of Q as elements and the conjuncts of Q as facts.

- Example:

  Conjunctive query Q :-- E(x,y),E(y,z),E(z,w)

  - Canonical instance I^Q consists of the facts E(x,y), E(y,z),E(z,w).
  - In other words, E^{I^Q} = {(x,y), (y,z), (z,w)}.

Canonical Database Instance

- Example:

  Conjunctive query Q(x,y) :-- E(x,z),E(z,y),P(z)

  or, equivalently,

  $$\{(x,y):\ \exists z(E(x,z) \wedge E(z,y) \wedge P(z))\}$$

  - Canonical instance I^Q consists of the facts E(x,z), E(z,y),P(z).
  - In other words, $E^{I^Q} = \{(x,z), (z,y)\}$ and $P^{I^Q}=\{z\}$

Canonical Conjunctive Queries and Canonical Instances

- Fact:
  - For every database instance I, we have that $I \models Q^I$.
  - For every Boolean conjunctive query Q, we have that $I^Q \models Q$.

- Fact: Let I be a database instance, let Q^I be its canonical conjunctive query and let I^{Q^I} be the canonical instance of Q^I. Then I is isomorphic to I^{Q^I}.

Canonical Conjunctive Queries and Canonical Instances

Magic Lemma: Assume that Q is a Boolean conjunctive query and J is a database instance. Then the following statements are equivalent.

- $J \models Q$.
- There is a homomorphism $h: I^Q \rightarrow J$.

Proof: Let Q be $\exists x_1 \ldots \exists x_m \, \varphi(x_1,\ldots,x_m)$.

1. $\Rightarrow$ 2. Assume that $J \models Q$. Hence, there are elements $a_1, \ldots, a_m$ in adom(J) such that $J \models \varphi(a_1,\ldots,a_m)$. The function h with $h(x_i) = a_i$, for $i=1,\ldots,m$, is a homomorphism from I^Q to J.

2. $\Rightarrow$ 1. Assume that there is a homomorphism $h: I^Q \rightarrow J$.
Then the values $h(x_i) = a_i$, for $i = 1,\ldots, m$, give values for the interpretation of the existential quantifiers $\exists x_i$ of Q in adom(J) so that $J \models \varphi(a_1,\ldots,a_m)$.

Homomorphisms, CQE, and CQC

The Homomorphism Theorem: Chandra & Merlin – 1977
For Boolean CQs Q and Q', the following are equivalent:

- $Q \subseteq Q'$
- There is a homomorphism h: $I^{Q'} \rightarrow I^{Q}$
- $I^{Q} \models Q'$.

In dual form:

The Homomorphism Theorem: Chandra & Merlin – 1977
For instances I and I', the following are equivalent:

- There is a homomorphism h: $I \rightarrow I'$
- $I' \models Q^{I}$
- $Q^{I'} \subseteq Q^{I}$

Homomorphisms, CQE, and CQC

The Homomorphism Theorem: Chandra & Merlin – 1977

For Boolean CQs Q and Q′, the following are equivalent:

1. $Q \subseteq Q'$
2. There is a homomorphism $h: I^{Q'} \rightarrow I^{Q}$
3. $I^{Q} \models Q'$.

Proof:

1. ⇒ 2. Assume $Q \subseteq Q'$. Since $I^{Q} \models Q$, we have that $I^{Q} \models Q'$.

Hence, by the Magic Lemma, there is a homomorphism from $I^{Q'}$ to I^{Q}.

2. ⇒ 3. It follows from the other direction of the Magic Lemma.

3. ⇒ 1. Assume that $I^{Q} \models Q'$. So, by the Magic Lemma, there is a homomorphism $h: I^{Q'} \rightarrow I^{Q}$. We have to show that if $J \models Q$, then $J \models Q'$. Well, if $J \models Q$, then (by the Magic Lemma), there is a homomorphism $h': I^{Q} \rightarrow J$. The composition $h' \circ h: I^{Q'} \rightarrow J$ is a homomorphism, hence
(once again by the Magic Lemma!), we have that $J \models Q'$.

Illustrating the Homomorphism Theorem

- **Example:**
  - Q: $\exists x_1 \exists x_2 \exists x_3 \exists x_4\ (E(x_1,x_2) \wedge E(x_2,x_3) \wedge E(x_3,x_4))$
  - Q′ : $\exists x_1 \exists x_2 \exists x_3\ (E(x_1,x_2) \wedge E(x_2,x_3))$

  Then:
- $Q \subseteq Q'$

  Homomorphism h: $I^{Q'} \rightarrow I^{Q}$ with

  $h(x_1) = x_1$, $h(x_2) = x_2$, $h(x_3) = x_3$.

- $Q' \nsubseteq Q$ (why?).

Illustrating the Homomorphism Theorem

- **Example:**
  - Q : $\exists x_1 \exists x_2 (E(x_1,x_2) \wedge E(x_2,x_1))$
  - Q′: $\exists x_1 \exists x_2 \exists x_3 \exists x_4 (E(x_1,x_2) \wedge E(x_2,x_1) \wedge E(x_2,x_3) \wedge E(x_3,x_2) \wedge$
    $E(x_3,x_4) \wedge E(x_4,x_3) \wedge E(x_4,x_1) \wedge E(x_1,x_4))$

  Then:
- $Q \subseteq Q'$

  Homomorphism h: $I^{Q'} \rightarrow I^{Q}$ with

  $h(x_1) = x_1$, $h(x_2) = x_2$, $h(x_3) = x_1$, $h(x_4) = x_2$.
- $Q' \subseteq Q$

  Homomorphism h′: $I^{Q} \rightarrow I^{Q'}$ with $h'(x_1) = x_1$, $h(x_2) = x_2$.
- Hence, $Q \equiv Q'$.

Illustrating the Homomorphism Theorem

Example: 3-Colorability

For a graph G=(V,E), the following are equivalent:

- G is 3-colorable
- There is a homomorphism h: $G \rightarrow K_3$
- $K_3 \models Q^G$
- $Q^{K_3} \subseteq Q^G$.

The Homomorphism Theorem for non-Boolean Conjunctive Queries

- So far, we have focused on Boolean conjunctive queries.
- However, the Homomorphism Theorem easily extends to conjunctive queries of arbitrary arities by considering homomorphisms that are the identity on the variables in the head of the conjunctive queries (written as rules).
- Moreover, the Homomorphism Theorem also extends to conjunctive queries with constants in some of the conjuncts.

  Find all cities one can reach by flying from San Jose with two stops

  $\{x: \exists x_1 \exists x_2\ (\text{FLIGHT}(\text{SanJose}, x_1) \wedge \text{FLIGHT}(x_1,x_2) \wedge \text{FLIGHT}(x_2,x))\}$.

The Homomorphism Theorem for non-Boolean Conjunctive Queries

The Homomorphism Theorem: Chandra & Merlin – 1977
Consider two k-ary conjunctive queries
$Q(x_1,\ldots,x_k)$:-- $R_1(\mathbf{u}_1), \ldots, R_n(\mathbf{u}_n)$ and $Q'(x_1,\ldots,x_k)$:-- $T_1(\mathbf{v}_1), \ldots, T_m(\mathbf{v}_m)$,
Then the following are equivalent:

- $Q \subseteq Q'$
- There is a homomorphism $h: I^{Q'} \rightarrow I^{Q}$ such that $h(x_1) = x_1, h(x_2) = x_2, \ldots, h(x_k) = x_k$.
- $I^{Q}, x_1,\ldots, x_k \models Q'$.

The Homomorphism Theorem for non-Boolean Conjunctive Queries

- Example: Consider the binary conjunctive queries

  Q(x,y):-- E(y,x),E(x,u)

  and

  Q'(x,y) :-- E(y,x), E(z,x),E(w,x),E(x,u)

Then $Q \subseteq Q'$ because there is a homomorphism

$h: I^{Q'} \rightarrow I^{Q}$ with h(x) =x and h(y) = y,

namely,

h(x) = x, h(y) = y, h(z) =y, h(w) = y, h(u) = u.

Combined complexity of CQC and CQE

Corollary: The following problems are NP-complete:

- Given two (Boolean) conjunctive queries Q and Q′ is $Q \subseteq Q'$?
- Given a Boolean conjunctive query Q and an instance I, does $I \models Q$?

Proof:

(a) Membership in NP follows from the Homomorphism Theorem:
$Q \subseteq Q'$ if and only if there is a homomorphism $h: I^{Q'} \rightarrow I^{Q}$

(b) NP-hardness follows from 3-Colorability:
G is 3-colorable if and only if $Q^{K_3} \subseteq Q^{G}$.

Conjunctive Query Equivalence

- The Conjunctive Query Equivalence Problem: Given two conjunctive queries Q and Q′, is $Q \equiv Q'$?

- Corollary: For conjunctive queries Q and Q′, we have that $Q \equiv Q'$ if and only if $I^Q \equiv_h I^{Q'}$.

- Corollary: The Conjunctive Query Equivalence Problem is NP-complete.
- Proof:
  - The following problem is NP-complete:
    Given a graph H containing a K_3, is H 3-colorable?
  - Let H be a graph containing a K_3. Then
    H is 3-colorable if and only if $Q^H \equiv Q^{K_3}$.

Combined Complexity vs. Data Complexity

Recall Vardi's Taxonomy of Query Evaluation:

- Combined Complexity: Both the query and the instance are part of the input.
- Data Complexity: Fix the query; the input consists of the instance only.

Complexity of Conjunctive Query Evaluation:

- The combined complexity of conjunctive query evaluation is NP-complete.
- The data complexity of conjunctive query evaluation is in P (in fact, it is in LOGSPACE).

The Complexity of Database Query Languages

	Relational Calculus	Conjunctive Queries
Query Evaluation Problem: Combined Complexity	PSPACE-complete	NP-complete
Query Evaluation Problem: Data Complexity	In LOGSPACE (hence, in P)	In LOGSPACE (hence, in P)
Query Equivalence Problem	Undecidable	NP-complete
Query Containment Problem	Undecidable	NP-complete

Conjunctive Query Minimization

- Definition: Let Q be a conjunctive query. A minimal equivalent conjunctive query to Q is a conjunctive query Q′ such that
  - Q is equivalent to Q′;
  - Q′ has as few conjuncts as any conjunct. query equivalent to Q.

- Example: Let Q be the conjunctive query

  Q(x,y) :-- E(y,x), E(z,x),E(w,x),E(x,u)

  Then the conjunctive query

  Q′(x,y):-- E(y,x),E(x,u)

  is a minimal equivalent conjunctive query to Q. (Why?)

Conjunctive Query Minimization and Conjunctive Query Evaluation

- A natural approach to conjunctive query evaluation is to first process a given conjunctive query Q, transform it to a minimal equivalent conjunctive query Q’, and then evaluate Q’ instead of Q.

- At first sight, this seems to be a promising approach.

- There is good news and bad news. First, the good news:

- Theorem: Let Q be a conjunctive query.
  - There is a minimal equivalent conjunctive query Q’ obtained from Q by removing zero or more conjuncts.
  - All minimal equivalent conjunctive queries to Q are isomorphic, i.e., their canonical instances are all isomorphic to each other.

  Proof: Use the Homomorphism Theorem (exercise).

Conjunctive Query Minimization and Conjunctive Query Evaluation

Next, the bad news:

Theorem:

- The following problem is NP-hard: Given two conjunctive queries Q and Q’, is Q’ a minimal equivalent query to Q?
- Consequently, unless P = NP, there is **no** polynomial-time algorithm such that, given a conjunctive query Q, the algorithm outputs a conjunctive query Q’ that is a minimal equivalent query to Q.

Proof: Reduction from 3-Colorability (exercise).

Note:

- This is not surprising since, as we saw, conjunctive query evaluation is NP-complete.
- There is an exponential-time algorithm such that, given a conjunctive query Q, the algorithm outputs a conjunctive query Q’ that is a minimal equivalent query to Q.

Tractable Cases of Conjunctive Query Evaluation

- Since conjunctive query evaluation is NP-complete, there has been an extensive investigation of special cases of conjunctive query evaluation for which the problem is in P.
- The key idea is to impose structural restrictions on the conjunctive queries considered:
  - Acyclic joins – M. Yannakakis (1981)
  - Various extensions of acyclicity have been studied over the years, including queries of bounded tree-width and queries of bounded hypertree width.
  - Extensive interaction with constraint satisfaction, logic, and graph theory.
- This belongs more to an advanced topics course or an independent study course.

Beyond Conjunctive Queries

- What can we say about query languages of intermediate expressive power between conjunctive queries and the full relational calculus?
- Conjunctive queries form the sublanguage of relational algebra obtained by using only cartesian product, projection, and selection with equality conditions.
- The next step would be to consider relational algebra expressions that also involve union.

Beyond Conjunctive Queries

- Definition:
  - A union of conjunctive queries is a query expressible by an expression of the form $q_1 \cup q_2 \cup \ldots \cup q_m$, where each q_i is a conjunctive query.
  - A monotone query is a query expressible by a relational algebra expression which uses only union, cartesian product, projection, and selection with equality condition.

- Fact:
  - Every union of conjunctive queries is a monotone query.
  - Every monotone query is equivalent to a union of conjunctive queries, but the union may have exponentially many disjuncts.

    (normal form for monotone queries).
  - Monotone queries are precisely the queries expressible by relational calculus expressions using $\wedge$, $\vee$, and $\exists$ only.

Unions of Conjunctive Queries and Monotone Queries

- Union of Conjunctive Queries

  $E \cup \pi_{1,4} (\sigma_{\$2=\$3} (E \times E))$ or, as a relational calculus expression,

  $E(x_1,x_2) \vee \exists z(E(x_1,z) \wedge E(z,x_2))$

- Monotone Query

  Consider the relation schemas $R_1(A,B)$, $R_2(A,B)$, $R_3(B,C)$, $R_4(B,C)$.

  The monotone query

  $(R_1 \cup R_2) \bowtie (R_3 \cup R_4)$

  is equivalent to the following union of conjunctive queries:

  $(R_1 \bowtie R_3) \cup (R_1 \bowtie R_4) \cup (R_2 \bowtie R_3) \cup (R_2 \bowtie R_4)$.

The Containment Problem for Unions of Conjunctive Queries

Theorem: Sagiv and Yannakakis – 1981

Let $q_1 \cup q_2 \cup \ldots \cup q_m$ and $q'_1 \cup q'_2 \cup \ldots \cup q'_n$ be two unions of conjunctive queries. Then the following are equivalent:

1. $q_1 \cup q_2 \cup \ldots \cup q_m \subseteq q'_1 \cup q'_2 \cup \ldots \cup q'_n$.
2. For every $i \leq m$, there is $j \leq n$ such that $q_i \subseteq q'_j$.

Proof: Use the Homomorphism Theorem

1. $\Rightarrow$ 2. Since $I^{q_i} \models q_i$, we have that $I^{q_i} \models q_1 \cup q_2 \cup \ldots \cup q_m$, hence $I^{q_i} \models q'_1 \cup q'_2 \cup \ldots \cup q'_n$, hence there is some $j \leq n$ such that $I^{q_i} \models q'_j$, hence (by the Homomorphism Theorem) $q_i \subseteq q'_j$.

2. $\Rightarrow$ 1. This direction is obvious.

The Containment Problem for Unions of Conjunctive Queries

- Corollary: The Query Containment Problem for Unions of Conjunctive Queries is NP-complete.
- Proof:
  - Membership in NP follows from the Sagiv-Yannakakis Theorem.
    - We guess m pairs (q'_{k_i}, h_{k_i}) and verify that for every $i \leq m$, the function h_{k_i} is a homomorphism from $I^{q'_{ki}}$ to I^{q_i}.
  - NP-hardness follows from the fact that Conjunctive Query Containment is a special case of this problem.

- Fact: The Query Evaluation Problem for Unions of Conjunctive Queries is NP-complete (combined complexity).

  Proof: Exercise.

The Complexity of Database Query Languages

	Relational Calculus	Conjunctive Queries	Unions of Conjunctive Queries
Query Evaluation: Combined Complexity	PSPACE-complete	NP-complete	NP-complete
Query Evaluation: Data Complexity	In LOGSPACE (hence, in P)	In LOGSPACE (hence, in P)	In LOGSPACE (hence, in P)
Query Equivalence	Undecidable	NP-complete	NP-complete
Query Containment	Undecidable	NP-complete	NP-complete

Monotone Queries

- Even though monotone queries have the same expressive power as unions of conjunctive queries, the containment problem for monotone queries has higher complexity than the containment problem for unions of conjunctive queries (syntax/complexity tradeoff)

- Theorem: Sagiv and Yannakakis – 1982
  The containment problem for monotone queries is Π_2^p-complete.

- Note:
  - Π_2^p is a complexity class that contains NP and is contained in PSPACE.
  - The prototypical Π_2^p-complete problem is $\forall\exists$-SAT, i.e., the restriction of QBF to formulas of the form
    $\forall x_1 \ldots \forall x_m \exists y_1 \ldots \exists y_n \varphi$.

The Complexity of Database Query Language

	Relational Calculus	Conjunctive Queries	Unions of Conjunctive Queries	Monotone Queries
Query Eval.: Combined Complexity	PSPACE-complete	NP-complete	NP-complete	NP-complete
Query Eval.: Data Complexity	In LOGSPACE (hence, in P)	In LOGSPACE (hence, in P)	In LOGSPACE (hence, in P)	In LOGSPACE (hence, in P)
Query Equivalence	Undecidable	NP-complete	NP-complete	Π_2^p-complete
Query Containment	Undecidable	NP-complete	NP-complete	Π_2^p-complete

Conjunctive Queries with Inequalities

- Definition: Conjunctive queries with inequalities form the sublanguage of relational algebra obtained by using only cartesian product, projection, and selection with equality and inequality ($\neq$, $<$, $\leq$) conditions.

- Example: $Q(x,y)$:-- $E(x,z)$, $E(z,w)$,$E(w,y)$, $z \neq w$, $z < y$.

- Theorem: (Klug – 1988, van der Meyden – 1992)
  - The query containment problem for conjunctive queries with inequalities is Π_2^p-complete.
  - The query evaluation problem for conjunctive queries with inequalities in NP-complete.

The Complexity of Database Query Languages

	Relational Calculus	Conjunctive Queries	Unions of Conjunctive Queries	Monotone Queries/ Conj.Queries with Inequal.
Query Eval.: Combined Complexity	PSPACE-complete	NP-complete	NP-complete	NP-complete
Query Eval.: Data Complexity	In LOGSPACE (hence, in P)	In LOGSPACE (hence, in P)	In LOGSPACE (hence, in P)	In LOGSPACE (hence, in P)
Query Equivalence	Undecidable	NP-complete	NP-complete	Π_2^p-complete
Query Containment	Undecidable	NP-complete	NP-complete	Π_2^p-complete

Conjunctive Queries with Inequalities

- Note: The Homomorphism Theorem **fails** for conjunctive queries with inequalities.
- Example: Consider the queries
  - q(x,y) :- E(x,y), F(u,v)
  - p(x,y) :- E(x,y), F(u,v), $u \neq v$.

  Then
  - q(x,y) $\not\subseteq$ p(x,y) (why?)
  - Yet, I^p is the same as I^q;
    in particular, there is a homomorphism h: $I^p \rightarrow I^q$.
  - Note also that p(x,y) $\subseteq$ q(x,y) (why?)

Complexity of Query Containment

- So, the complexity of query containment for conjunctive queries and their variants is well understood.

Caveat:

- All preceding results assume **set semantics**, i.e., queries take **sets** as inputs and return **sets** as output (duplicates **are** eliminated).
- DBMS, however, use **bag semantics (multiset semantics)**, since they return **bags (multisets)**

  (recall that duplicates are **not** eliminated in SQL, unless explicitly specified).

A *Real* Conjunctive Query

- Consider the following SQL query:

  Table Employee has attributes salary, dept, …

```
SELECT salary
FROM   Employee
WHERE  dept = 'CS'
```

- Recall that SQL keeps duplicates, because:
  - User may care about duplicates

    {100, 100, 200} different than {100, 200} for **AVERAGE**
  - In general, bags can be more "efficient" than sets.

Query Evaluation under Bag Semantics

Operation	Multiplicity
Union $R_1 \cup R_2$	$m_1 + m_2$
Intersection $R_1 \cap R_2$	$\min(m_1, m_2)$
Product $R_1 \times R_2$	$m_1 \times m_2$
Projection and Selection	Duplicates are not eliminated

- R_1

A	B
1	2
1	2
2	3

- R_2

B	C
2	4
2	5

- $(R_1 \bowtie R_2)$

A	B	C
1	2	4
1	2	4
1	2	5
1	2	5

Bag (Multiset) Semantics

S. Chaudhuri & M.Y. Vardi – 1993

Optimization of ***Real*** Conjunctive Queries

- Called for a re-examination of conjunctive-query optimization under bag semantics.

- In particular, they initiated the study of the containment problem for conjunctive queries under bag semantics.

Bag Semantics vs. Set Semantics

- $Q^{BAG}(I)$: Result of evaluating Q on (bag) database instance I.
- For bags R_1, R_2:

  $R_1 \subseteq_{BAG} R_2$ if $m(\mathbf{a},R_1) \leq m(\mathbf{a},R_2)$, for every tuple $\mathbf{a}$.
- $Q_1 \subseteq_{BAG} Q_2$ if for every (bag) database I, we have that

$$Q_1^{BAG}(I) \subseteq_{BAG} Q_2^{BAG}(I).$$

Fact:

- $Q_1 \subseteq_{BAG} Q_2$ implies $Q_1 \subseteq Q_2$ (this is obvious from the definitions).

- The converse does **not** always hold.

Bag Semantics vs. Set Semantics

Fact: $Q_1 \subseteq Q_2$ need not imply that $Q_1 \subseteq_{BAG} Q_2$.

Example:

- $Q_1(x)$:- $P(x), T(x)$
- $Q_2(x)$:- $P(x)$

- $Q_1 \subseteq Q_2$ (this is obvious from the definitions)
- $Q_1 \not\subseteq_{BAG} Q_2$
- Consider the (bag) instance I = {P(a), T(a), T(a)}.
  Then:
  - $Q_1(I) = \{a,a\}$
  - $Q_2(I) = \{a\}$, so $Q_1(I) \not\subseteq Q_2(I)$.

Conjunctive Query Evaluation under Bag Semantics

- Boolean query

  Q :- E(x,y), E(y,z), E(z,x)

- **Set Semantics**

  Q(G) = "true" if and only if the graph G contains a triangle.

- **Bag Semantics**

  $Q^{BAG}(G)$ = # of triangles in the graph G.

Conjunctive Query Evaluation under Bag Semantics

Consider:

- K_3: the complete graph with 3 nodes
- $G=(V,E)$ an arbitrary graph and the canonical conjunctive query Q^G of G.

Then

- **Set Semantics**

  $Q^G(K_3)$ = "true" if and only if G is 3-colorable.
- **Bag Semantics**

  $Q^{G\text{-}BAG}(K_3)$ = # 3-colorings of the graph G.

Corollary: The conjunctive query evaluation problem under bag semantics is #P-complete.

Query Containment under Bag Semantics

- Chaudhuri & Vardi - 1993 stated that:

  Under bag semantics, the containment problem for conjunctive queries is Π_2^p-hard.

- **Open Problem:**
  - What is the **exact complexity**, under bag semantics, of the containment problem for conjunctive queries?
  - Is this problem **decidable**? Even this is not known to date!

Unions of Conjunctive Queries

Theorem: Ioannidis & Ramakrishnan – 1995
Under bag semantics, the containment problem for unions of conjunctive queries is **undecidable**.

Hint of Proof:
Reduction from
Hilbert's 10th Problem.

Hilbert's 10th Problem

- Hilbert's 10th Problem – 1900
  (10th in his list of 23 problems)
  Find an algorithm for the following problem:
  Given a polynomial equation $p(x_1,...,x_n) = 0$ with integer coefficients, does it have an all-integer solution?

- Matijasevic – 1971
  - Hilbert's 10th Problem is **undecidable**, hence **no** such algorithm exists.
  - **Undecidable**, even for degree $d = 4$ and $n = 58$.

Hilbert's 10th Problem

- Fact: The following variant of Hilbert's 10th Problem is undecidable:
  - Given two polynomials $p_1(x_1,\ldots x_n)$ and $p_2(x_1,\ldots x_n)$ with positive integer coefficients and no constant terms, is it true that $p_1 \leq p_2$? i.e., is it true that $p_1(a_1,\ldots,a_n) \leq p_2(a_1,\ldots a_n)$, for all positive integers $a_1,\ldots,a_n$?
- So, there is no algorithm for deciding questions like:
  - Is $3x_1^4x_2x_3 + 2x_2x_3 \leq x_1^6 + 5x_2x_3$?

Unions of Conjunctive Queries

Theorem: Ioannidis & Ramakrishnan – 1995
Under bag semantics, the containment problem for unions of conjunctive queries is **undecidable**, even if all relations are unary.

Hint of Proof:

- Reduction from the previous variant of Hilbert's 10th Problem:
  - Use joins of unary relations to encode monomials (products of variables).
  - Use unions to encode sums of monomials.

Unions of Conjunctive Queries

Theorem: Ioannidis & Ramakrishnan – 1995
Under bag semantics, the containment problem for unions of conjunctive queries is **undecidable**, even if all relations are unary.

Example: Consider the polynomial $3x_1^4x_2x_3 + 2x_2x_3$

- The monomial $x_1^4x_2x_3$ is encoded by the conjunctive query $P_1(w),P_1(w),P_1(w), P_1(w), P_2(w),P_3(w)$.
- The monomial x_2x_3 is encoded by the conjunctive query $P_2(w),P_3(w)$.
- The polynomial $3x_1^4x_2x_3 + 2x_2x_3$ is encoded by the union having:
  - three copies of $P_1(w),P_1(w),P_1(w), P_1(w), P_2(w),P_3(w)$ and
  - two copies of $P_2(w),P_3(w)$.

Computational Complexity of Query Containment

Class of Queries	**Complexity – Set Semantics**	**Complexity – Bag Semantics**
Conjunctive queries	NP-complete Chandra Merlin – 1977	**Open**
Unions of conj. queries	NP-complete Sagiv-Yannakakis – 1981	Undecidable Ioannidis-Ramakrishnan - 1995
Conj. queries with $\neq$, $\leq$, $\geq$	Π_2^p-complete Van der Meyden – 1992	
Relational calculus queries	Undecidable Trakhtenbrot - 1949	Undecidable

Bag Semantics and Conjunctive Queries with $\neq$

Theorem: Jayram, K… , Vee – 2006
Under bag semantics, the containment problem for conjunctive queries with $\neq$ is **undecidable**.

In fact, this problem is **undecidable** even if

- the queries use only a single relation of arity 2;
- the number of inequalities in the queries is at most some fixed constant.

Bag Semantics and Conjunctive Queries with $\neq$

Proof Idea:

Reduction from another variant of Hilbert's10th Problem:

Given homogeneous polynomials
$P_1(x_1,\ldots,x_{59})$ and $P_2(x_1,\ldots,x_{59})$
both with integer coefficients and both of degree 5,
is $P_1(x_1,\ldots,x_{59}) \leq (x_1)^5 P_2(x_1,\ldots,x_{59})$,
for all integers $x_1,\ldots,x_{59}$?

The reduction is much more involved than the earlier reduction for unions of conjunctive queries.

Proof Idea (continued)

- Given polynomials P_1 and P_2
  - Both with integer coefficients
  - Both homogeneous, degree 5
  - Both with at most $n=59$ variables
- We want to find Q_1 and Q_2 such that
  - Q_1 and Q_2 are conjunctive queries with inequalities $\neq$
  - $P_1(x_1,\ldots, x_{59}) \leq (x_1)^5 P_2(x_1,\ldots, x_{59})$

    for all integers $x_1, \ldots, x_{59}$

    if and only if

    $Q_1(D) \subseteq_{BAG} Q_2(D)$ for all (bag) databases D.

Proof Outline:

Proof is carried out in three steps.

Step 1: Only consider DBs of a special form.
Show how to use conjunctive queries to encode polynomials (without using inequalities!)

Step 2: "Force" DB to have special form using inequalities.

- If D is not of special form, then
  $Q_1(D) \subseteq_{BAG} Q_2(D)$ necessarily.

Step 3: Show that we only need a single relation of arity 2.

Step 1: DBs of a Special Form - Example

- Encode a homogeneous, 2-variable, degree 2 polynomial in which all coefficients are 1.

  $$P(x_1,x_2) = x_1^2 + x_1x_2 + x_2^2$$

- DBs of special form:
  - Ternary relation TERM consisting of
    - (X_1,X_1,T_1), (X_1,X_2,T_2), (X_2,X_2,T_3)

    all special DBs have precisely this table for TERM
  - Binary relation VALUE
    - Table for VALUE varies to encode different values for the variables x_1, x_2.
- Query Q :- $\text{TERM}(u_1,u_2,t)$, $\text{VALUE}(u_1,v_1)$, $\text{VALUE}(u_2,v_2)$

Step 1: DBs of a Special Form - Example

- $P(x_1,x_2) = x_1^2 + x_1x_2 + x_2^2$
  $x_1 = 3$, $x_2 = 2$, $P(3,2) = 3^2 + 3\cdot 2 + 2^2 = 19$.
- Query Q :- $TERM(u_1,u_2,t)$, $VALUE(u_1,v_1)$, $VALUE(u_2,v_2)$
- DB D of special form:
  - TERM: (X_1,X_1,T_1), (X_1,X_2,T_2), (X_2,X_2,T_3)
  - VALUE: $(X_1,1)$, $(X_1,2)$, $(X_1,3)$
    $(X_2,1)$, $(X_2,2)$

Claim: $P(3,2) = 19 = Q^{BAG}(D)$

Step 1: DBs of a Special Form - Example

- $P(3,2) = 3^2 + 3\cdot 2 + 2^2 = 19$.
- Query Q :- $TERM(u_1,u_2,t)$, $VALUE(u_1,v_1)$, $VALUE(u_2,v_2)$
- D has TERM: (X_1,X_1,T_1), (X_1,X_2,T_2), (X_2,X_2,T_3)
  VALUE: $(X_1,1)$, $(X_1,2)$, $(X_1,3)$, $(X_2,1)$, $(X_2,2)$
- $Q^{BAG}(D) = 19$, because:
  - $t \rightarrow T_1$, $u_1 \rightarrow X_1$, $u_2 \rightarrow X_1$. Hence:
    $v_1 \rightarrow 1,2$, or 3 and $v_2 \rightarrow 1$ or 2, so we get 3^2 witnesses.
  - $t \rightarrow T_2$, $u_1 \rightarrow X_1$, $u_2 \rightarrow X_2$. Hence:
    $v_1 \rightarrow 1,2$, or 3 and $v_2 \rightarrow 1$ or 2, so we get $3\cdot 2$ witnesses.
  - $t \rightarrow T_3$, $u_1 \rightarrow X_2$, $u_2 \rightarrow X_2$. Hence:
    $v_1 \rightarrow 1$ or 2, and $v_2 \rightarrow 1$ or 2, so we get 2^2 witnesses.

Step 1: Complete Argument and Wrap-up

- Previous technique only works if all coefficients are 1
- For the complete argument:
  - add a fixed table for every term to the DB;
  - encode coefficients in the query;
  - only table for VALUE can vary.
- **Summary:**
  - If the database has a special form, then we can encode separately homogeneous polynomials P_1 and P_2 by conjunctive queries Q_1 and Q_2.
  - By varying table for VALUE, we vary the variable values.
  - **No** $\neq$-constraints are used in this encoding; hence, conjunctive query containment is **undecidable**, if restricted to databases of the special form.

Step 2: Arbitrary Databases

Idea:

Use inequalities $\neq$ in the encoding queries
to achieve the following:

- If a database D is of special form, then we are back to the previous case.
- If a database D is not of special form, then
  $Q_1(D) \subseteq_{BAG} Q_2(D)$ necessarily.

Step 2: Arbitrary Databases - Hint

1. Ensure that certain “facts” in special-form DBs appear (else neither query is satisfied).

- This is done by adding a part of the canonical query of special-form DBs as subgoals to each encoding query.

2. Modify special-form DBs by adding gadget tuples to TERM and to VALUE.

- TERM: (X_1,X_1,T_1), (X_1,X_2,T_2), (X_2,X_2,T_3), (T_0,T_0,T_0)
- VALUE: $(X_1,1)$, $(X_1,2)$, $(X_1,3)$, $(X_2,1)$, $(X_2,2)$, (T_0,T_0)

3. Add extra subgoals to Q_2, so that if D is not of special form, then Q_2 “benefits” more than Q_1 and, as a result, $Q_1(D) \subseteq_{BAG} Q_2(D)$.

Step 2: Arbitrary Databases - Example

- $P_1(x_1,x_2) = x_1^2 + x_1x_2 + x_2^2$
- $Poly_1(u_1,u_2,t)$:- $TERM(u_1,u_2,t), VALUE(u_1,v_1), VALUE(u_2,v_2)$
  the query encoding P_1 on special-form DBs.
  - TERM: $(X_1,X_1,T_1), (X_1,X_2,T_2), (X_2,X_2,T_3), (T_0,T_0,T_0)$
  - VALUE: $(X_1,1), (X_1,2), (X_1,3), (X_2,1), (X_2,2), (T_0, T_0)$

- Q_1 :- $Poly_1(u_1,u_2,t)$
- Q_2 :- $Poly_2(u_1, u_2, t), Poly_1(w_1, w_2, w), w \neq T_1, w \neq T_2, w \neq T_3$

Fact:

- If DB is of special form, then Q_2 gets no advantage, because $w \to T_0, w_1 \to T_0, w_2 \to T_0$ is the only possible assignment.
- If DB not of special form, say it has an extra fact (X_2,X_1,T'), then both Q_1 and Q_2 can use it equally.

Step 2: Arbitrary Databases – Wrap-up

- Additional tricks are needed for the full construction.
- Full construction uses seven different control gadgets.
  - Additional complications when we encode coefficients.
  - Inequalities $\neq$ are used in both queries.
- Number of inequalities $\neq$ depends on size of special-form DBs, not counting the facts in VALUE table.
  - Hence, depends on degree of polynomials, # of variables.
  - It is a huge constant (about 59^{10}).

Complexity of Query Containment

Class of Queries	**Complexity – Set Semantics**	**Complexity – Bag Semantics**
Conjunctive queries	NP-complete CM – 1977	**Open**
Unions of conj. queries	NP-complete SY - 1980	Undecidable IR - 1995
Conj. queries with $\neq$, $\leq$, $\geq$	Π_2^p-complete vdM - 1992	Undecidable JKV - 2006
First-order (SQL) queries	Undecidable Gödel - 1931	Undecidable

Directions for Future Work

- **Major Open Problem:**
  Conjunctive query containment problem (no inequalities), under bag semantics.
  - Is it decidable?
  - If so, what is the exact complexity?

- Identify classes of queries for which, under bag semantics, the containment problem is tractable.

- Pinpoint the exact complexity of bag-equivalence.

The Query Equivalence Problem

- Query Equivalence: given Q_1, Q_2, is $Q_1 \equiv Q_2$?
- Under bag semantics,
  - For conjunctive queries, it has the same complexity as GRAPH ISOMORPHISM

    Chaudhuri & Vardi - 1993
  - For conjunctive queries with inequalities $\neq$,
    - Lower Bound: It is GRAPH ISOMORPHISM-hard.
    - Upper Bound: It is in PSPACE

      Nutt, Sagin, Shurin (Cohen) – 1998

    Big gap between the lower and the upper bound.

Limitations of Relational Algebra & Relational Calculus

Outline:

- Relational Algebra and Relational Calculus have substantial expressive power. In particular, they **can** express
  - Natural Join
  - Quotient
  - Unions of conjunctive queries
  - …
- However, they **cannot** express recursive queries.
- Datalog is a declarative database query language that augments the language of conjunctive queries with a recursion mechanism.

Parents, Grandparents, and Greatgrandparents

- Let PARENT be a binary relational schema such that if (a,b) $\in$ PARENT in some database instance, then a is a parent of b.

- Using PARENT, we can define GRANDPARENT and GREATGRANPARENT by the conjunctive queries:

  GRANDPARENT(x,y) :- PARENT(x,z), PARENT(z,y)

  and

  GREATGRANDPARENT(x,y) :- PARENT(x,z), PARENT(z,w), PARENT(w,y)

- Similarly, we can define GREATGREATGRANPARENT by a conjunctive query, and so on up to any fixed level of ancenstry.

Parents and Ancestors

- Question: Is there a relational algebra (relational calculus) expression that defines ANCESTOR from PARENT?
- Note: This type of question occurs in other related concepts:
  - Given a binary relation MANAGES(manager, employee), is there a relational algebra (relational calculus) expression that defines HIGHER-MANAGER
  - Given a binary relation DIRECT(from,to) about flights, is there a relational algebra (relational calculus) expression that defines CAN-FLY(from,to)?
  - More abstractly, given a binary relation E, is there a relational algebra (relational calculus) expression that defines the Transitive Closure TC of E?

Edges and Paths

Definition: Let E be a binary relation

- For every $n \geq 1$, let PATH_n be the binary query:

  "given a and b, is there a path of length n from a to b along edges from E?"
- PATH is the binary query:

  "given a and b, is there a path from a to b along edges from E?"

Fact:

- For every $n \geq 1$, the query PATH_n is expressible by a conjunctive query. (Why?)
- Hence, PATH is expressible by an infinite union of conjunctive queries:

$$\text{PATH} \equiv \text{PATH}_1 \cup \text{PATH}_2 \cup \ldots \cup \text{PATH}_n \cup \ldots$$

Edges, Paths, and Transitive Closure

Facts: Let E be a binary relation

- PATH is the Transitive Closure of E, i.e.,

  the smallest binary relation T such that
  - $E \subseteq T$
  - T is transitive (if $(a,b) \in T$ and $(b,c) \in T$, then $(a,c) \in T$).

- There are several well-known efficient algorithms for computing the Transitive Closure of a given binary relation E
  - Floyd–Warshall Algorithm (taught in CMPS 101).

- Recall that the following problem is NLOGSPACE-complete:

  Given E, a and b, is there a path from a to b? (i.e., is $(a,b) \in TC$?)

Transitive Closure and Relational Calculus

- Question: Is there a relational algebra (relational calculus) expression that defines ANCESTOR from PARENT? In other words, is there a relational algebra (relational calculus) expression that defines the Transitive Closure of a given binary relation E?

- Theorem: A. Aho and J. Ullman – 1979

  There is **no** relational algebra (or relational calculus) expression that defines the Transitive Closure of a given binary relation E.

Transitive Closure and Relational Calculus

Theorem: A. Aho and J. Ullman – 1979
There is **no** relational algebra (or relational calculus) expression that defines the Transitive Closure of a given binary relation E.

Note:

- The proof of this result requires methods from mathematical logic.
- Ehrenfeucht-Fraïssé Games is a powerful method for proving limitations in the expressive power of relational calculus.
  - Two-person perfect-information combinatorial games in which two players take turns and pick elements in two different database instances.
  - One player tries to maintain a partial isomorphism between the moves played; the other player tries to violate this.

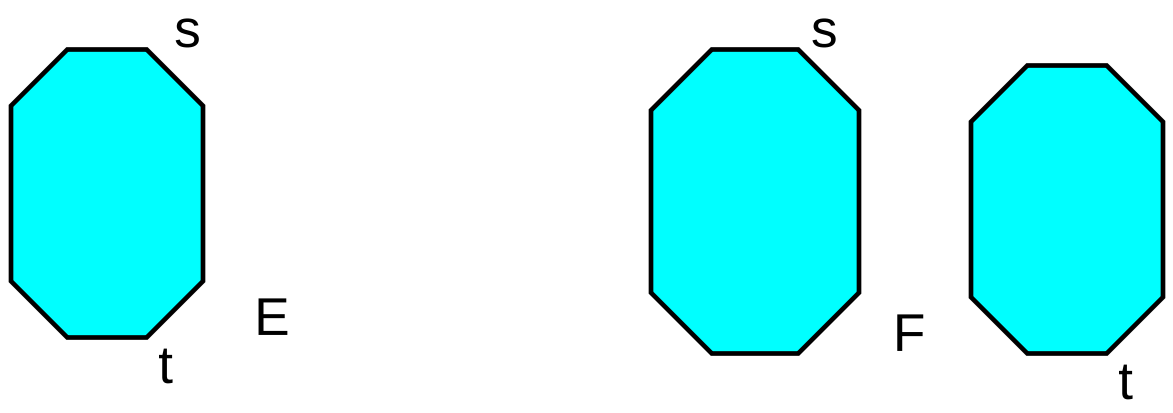
Transitive Closure and Relational Calculus

Theorem: A. Aho and J. Ullman – 1979

There is **no** relational algebra (or relational calculus) expression that defines the Transitive Closure of a given binary relation E.

Intuition behind this result:

- Relational Calculus queries can only express “local” properties



Limitations of Relational Calculus and Relational Calculus

- There is **no** relational algebra (relational calculus) expression involving PARENT that defines ANCESTOR.
- ANCESTOR is definable by an **infinite union** of conjunctive queries, but is not definable by any **finite union** of conjunctive queries.
- Aho and Ullman's Theorem reveals a **limitation** in the expressive power of relational algebra and relational calculus, namely
  - They **cannot** express recursive queries.

Overcoming the Limitations of Relational Calculus

- Question: What is to be done to overcome the limitations of the expressive power of relational calculus?
- Answer 1: Embedded Relational Calculus (Embedded SQL):
  - Allow SQL commands inside a conventional programming language, such as C, Java, etc.
  - This is an inferior solution, as it destroys the high-level character of SQL.
- Answer 2:
  - Augment relational calculus with a high-level declarative mechanism for recursion.
  - Conceptually, this a superior solution as it maintains the high-level declarative character of relational calculus.

Datalog

- Datalog = "Conjunctive Queries + Recursion"
- Datalog was introduced by Chandra and Harel in 1982 and has been studied by the research community in depth since that time:
  - Hundreds of research papers in major database conferences;
  - Numerous doctoral dissertations.
  - Recent applications outside databases:
    - Specification of network properties
    - Access control languages
- SQL:1999 and subsequent versions of the SQL standard provide support for a sublanguage of Datalog, called linear Datalog.

Datalog Syntax

- Definition: A Datalog program π is a finite set of rules each expressing a conjunctive query

$$T(x_1,\ldots,x_k) \text{ :-- } R_1(\mathbf{u}_1), \ldots, R_n(\mathbf{u}_n),$$

where each variable x_i occurs in the body of the rule.

- Some relational symbols occurring in the heads of the rules may also occur in the bodies of the rules (unlike the rules for conjunctive queries).
  - These relational symbols are the recursive relational symbols; they are also known as intensional database predicates (IDBs).
- The remaining relational symbols in the rules are known as the extensional database predicates (EDBs).

Datalog

- Example: Datalog program for Transitive Closure

  T(x,y) :- E(x,y)

  T(x,y) :- E(x,z), T(z,y)

  - E is the EDB predicate
  - T is the IDB predicate
  - The intuition is that the Datalog program gives a recursive specification of the IDB predicate T in terms of the EDB E.

- Example: Another Datalog program for Transitive Closure

  T(x,y) :- E(x,y)

  T(x,y) :- T(x,z), T(z,y)

  (“divide and conquer” algorithm for Transitive Closure)

Datalog

- Example: Paths of Even and Odd Length

  Consider the Datalog program:

  ```
  ODD(x,y)  :-  E(x,y)
  ODD(x,y)  :-  E(x,z), EVEN(z,y)
  EVEN(x,y) :-  E(x,z), ODD(z,y).
  ```

  - E is the EDB predicate
  - EVEN and ODD are the IDB predicates.
  - So, a Datalog program may have several different IDB predicates (and it may have several different EDB predicates as well).
  - This program gives a recursive specification of the IDB predicates EVEN and ODD in terms of the EDB predicate E.
  - This is a Datalog program expressing mutual recursion.

Conjunctive Queries vs. Datalog

- As we have seen, conjunctive queries can be written as rules:

  $T(x_1,\ldots,x_k)$:-- $R_1(\mathbf{u}_1)$, …, $R_n(\mathbf{u}_n)$.

  - In such a rule, the relation symbol in the head **does not occur** in the body of the rule.
- Datalog programs are finite sets of rules.
  - In a Datalog program, however, a relation symbols occurring in the heads of a rule:
    - **may also occur** in the body of the same rule

      T(x,y) :- E(x,z), T(z,y)

    - or, it **may occur** in the body of another rule in the program

      ODD(x,y) :- E(x,z), EVEN(z,y)

      EVEN(x,y) :- E(x,z), ODD(z,y).

Datalog Semantics

- Question: What is the precise semantics of a Datalog program?
- Answer: Datalog programs can be given two different types of semantics.
  - Declarative Semantics (denotational semantics)
    - Smallest solutions of recursive specifications.
    - Least fixed-points of monotone operators.
  - Procedural Semantics (operational semantics)
    - An iterative process for computing the "meaning" of Datalog programs.
- Main Result: The declarative semantics coincides with the procedural semantics.

Declarative Semantics of Datalog Programs

Motivation:

- Recall the recursive definition of the factorial function $f(n) = n!$

  $$f(0) = 1$$
  $$f(n+1) = (n+1) \cdot f(n)$$

  - These two equations give a recursive specification of the factorial function $f(n) = n!$
  - The factorial function is the only function on the integers that satisfies this specification.
- Similarly, recall the recursive definition of $g(x,y) = x^y$

  $$g(x,0)= 1$$
  $$g(x,y+1)= x \cdot g(x,y)$$

  - The exponential function $g(x,y) = x^y$ is the only function on the integers that satisfies this specification.

Declarative Semantics of Datalog Programs

- Each Datalog program can be viewed as a recursive specification of its IDB predicates.
- This specification is expressed using relational algebra operators
  - The body of each rule uses π, σ, and cartesian product $\times$
  - All rules having the same predicate in the head are combined using union.
  - The recursive specification is given by equations involving unions of conjunctive queries.

- Example: T(x,y) :- E(x,y)
  T(x,y):- T(x,z), T(z,y)
  - Recursive equation:

$$T = E \cup \pi_{1,4}(\sigma_{\$2=\$3}(T \times T))$$

Declarative Semantics of Datalog Programs

Example: Consider the Datalog program:

```
ODD(x,y)  :-  E(x,y)
ODD(x,y)  :-  E(x,z), EVEN(z,y)
EVEN(x,y) :-  E(x,z), ODD(z,y).
```

- System of recursive equations:

$$\text{ODD} = \text{E} \cup \pi_{1,4}(\sigma_{\$2=\$3}\ (\text{E}\times\text{EVEN}))$$

$$\text{EVEN} = \pi_{1,4}(\sigma_{\$2=\$3}\ (\text{E}\times\text{ODD})).$$

Declarative Semantics of Datalog Programs

- Unlike the recursive equations for the factorial and the exponential function, recursive equations arising from Datalog programs **need not** have a unique solution.

- Example: Consider the recursive equation:

$$T \;=\; E \cup \pi_{1,4}(\sigma_{\$2=\$3}\,(T \times T))$$

Let $E = \{ (1,2), (2,3) \}$.

Then both T_1 and T_2 satisfy this recursive equation, where

  - $T_1 = \{ (1,2), (2,3), (1,3) \}$
  - $T_2 = \{ (1,2),(2,1),(2,3),(3,2),(1,3),(3,2),(1,1),(2,2),(3,3) \}$.

Furthermore, this recursive equation has many other solutions.

Declarative Semantics of Datalog Programs

- Theorem: Every recursive equation arising from a Datalog program has a smallest solution (smallest w.r.t. the $\subseteq$ partial order).
- Example: Datalog program

  T(x,y) :- E(x,y)

  T(x,y):- T(x,z), T(z,y)

  - Recursive equation:

    $$T = E \cup \pi_{1,4}(\sigma_{\$2=\$3}(T \times T))$$

  - The smallest solution of this recursive equation is the Transitive Closure of E.

- Note: This is a special case of the Knaster-Tarski Theorem for smallest solutions of recursive equations arising from monotone operators (it is important that Datalog uses monotone relational algebra operators only).

Declarative Semantics of Datalog Programs

- Theorem: Every recursive equation arising from a Datalog program has a smallest solution (smallest w.r.t. the $\subseteq$ partial order).
- Definition: The (declarative) semantics of a Datalog program is the smallest solution of the system of recursive equations arising from the Datalog program.
- Note:
  - If the Datalog program has more than one IDBs, then we get a system of recursive equations, instead of single one.
- Question:
  - What does "smallest solution of a system" mean in this case?

Declarative Semantics of Datalog Programs

Example: Consider again the Datalog program:

```
ODD(x,y)  :-  E(x,y)
ODD(x,y)  :-  E(x,z), EVEN(z,y)
EVEN(x,y) :-  E(x,z), ODD(z,y).
```

- System of recursive equations:

  $$\mathrm{ODD} = E \cup \pi_{1,4}(\sigma_{\$2=\$3}\,(E \times \mathrm{EVEN}))$$

  $$\mathrm{EVEN} = \pi_{1,4}(\sigma_{\$2=\$3}\,(E \times \mathrm{ODD})).$$

- The smallest solution of this system is a pair of relations (P,Q) such that
  1. (P,Q) satisfies this system (e.g., $Q = \pi_{1,4}(\sigma_{\$2=\$3}(E \times Q))$).
  2. If a pair (P’,Q’) satisfies this system, then $P \subseteq P'$ and $Q \subseteq Q'$.

Declarative Semantics of Datalog Programs

- Question:
  - How difficult is it to compute the declarative semantics of Datalog programs?
  - In other words, what is the computational complexity of the query evaluation problem for Datalog queries?
- Note: On the face of their definition, the declarative semantics of Datalog programs do not give rise to an algorithm for computing this semantics.

Procedural Semantics of Datalog Programs

Definition: Let π be a Datalog program. The procedural semantics of π are obtained by the following bottom-up evaluation of the recursive predicates (IDBs) of π:

1. Set all IDBs of π to $\emptyset$.
2. Apply all rules of π in parallel; update the IDBs by evaluating the bodies of the rules.
3. Repeat until no IDB predicate changes.
4. Return the values of the IDB predicates obtained at the end of Step 3.

Procedural Semantics of Datalog Programs

Example: Datalog program for Transitive Closure

```
T(x,y) :- E(x,y)
T(x,y):-  E(x,z),T(z,y)
```

- Bottom-up evaluation:

  $T^0 = \emptyset$

  $T^{n+1} = \{(a,b): E(a,b) \vee \exists z(E(a,z) \wedge T^n(z,b))\}$

Fact: The following statements are true:

- $T^n = \{ (a,b)$: there is a path of length at most n from a to b $\}$
- Transitive Closure of E = $\cup_{n \geq 1} T^n$.

Proof: By induction on n.

Procedural Semantics of Datalog Programs

Example: Another Datalog program for Transitive Closure

$T(x,y)$:- $E(x,y)$

$T(x,y)$:- $T(x,z),T(z,y)$

- Bottom-up evaluation:

$$T^0 = \varnothing$$

$$T^{n+1} = \{(a,b): E(a,b) \vee \exists z(T^n(a,z) \wedge T^n(z,b))\}$$

Fact: The following statements are true:

- $T^n = \{ (a,b)$: there is a path of length at most 2^n from a to b $\}$
- Transitive Closure of E = $\cup_{n \geq 1} T^n$.

Proof: By induction on n.

Procedural Semantics of Datalog Programs

Example: Consider the Datalog program

```
ODD(x,y)   :-  E(x,y)
ODD(x,y)   :-  E(x,z), EVEN(z,y)
EVEN(x,y)  :-  E(x,z), ODD(z,y)
```

- Bottom-up evaluation:

$$\mathrm{ODD}^0 = \emptyset$$
$$\mathrm{EVEN}^0 = \emptyset$$
$$\mathrm{ODD}^{n+1} = \{(a,b): E(a,b) \vee \exists z(E(a,z) \wedge \mathrm{EVEN}^n(z,b))\}$$
$$\mathrm{EVEN}^{n+1} = \{(a,b): \exists z(E(a,z) \wedge \mathrm{ODD}^n(z,b))\}$$

Fact: The following statements are true:

- $\cup_{n \geq 1} \mathrm{ODD}^n$ = { (a,b): there is a path of odd length from a to b }
- $\cup_{n \geq 1} \mathrm{EVEN}^n$ = { (a,b): there is a path of even length from a to b }.

Proof: By induction on n.

Declarative vs. Procedural Datalog Semantics

Theorem: Let π be a Datalog program. Then the following are true:

- The bottom-up evaluation of the procedural semantics of π terminates within a number of steps bounded by a polynomial in the size of the database instance (= size of the EDB predicates).
- The declarative semantics of π coincides with the procedural semantics of π.

Proof: For simplicity, assume that π has a single IDB T of arity k.

- By induction on n, show that $T^n \subseteq T^{n+1}$, for every n.

  (this uses the monotonicity of unions of conjunctive queries).
- Hence, $T^0 \subseteq T^1 \subseteq \ldots \subseteq T^n \subseteq T^{n+1} \subseteq \ldots$
- Since each $T^n \subseteq adom(I)^k$, there is an $m \leq |adom(I)|^k$ such that $T^m = T^{m+1}$.

Declarative vs. Procedural Datalog Semantics

Theorem: Let π be a Datalog program. Then the following are true:

- The bottom-up evaluation of the procedural semantics of π terminates within a number of steps bounded by a polynomial in the size of the database instance (= size of the EDB predicates).
- The declarative semantics of π coincides with the procedural semantics of π.

Proof: For simplicity, assume that π has a single IDB T of arity k.

- Since $T^m = T^{m+1}$, we have that the procedural semantics produces a solution to the recursive equation arising from π.
- By induction on n, show that if T* is another solution of this recursive equation, then $T^n \subseteq T^*$, for all n

  (use the monotonicity of unions of conjunctive queries again).
- In particular, $T^m \subseteq T^*$, hence T^m is the smallest solution of this recursive equation.

The Query Evaluation Problem for Datalog

Theorem: Let π be a Datalog program. There is a polynomial-time algorithm such that, given a database instance I, it evaluates π on I (i.e., it computes the semantics of π on I).

Proof: The bottom-up evaluation of the procedural semantics of π runs in polynomial time because:

- The number of iterations is bounded by a polynomial in the size of I.
- Each step of the iteration can be carried out in polynomial time (why?).

Corollary: The data complexity of Datalog is in P.

The Query Evaluation Problem for Datalog

Corollary: The data complexity of Datalog is in P.

Theorem: The combined complexity of Datalog is EXPTIME-complete.

Note:

- $P \subseteq NP \subseteq PSPACE \subseteq EXPTIME$.
- Moreover, it is known that P is properly contained in EXPTIME.
- Thus, Datalog has higher combined complexity than relational calculus (since the combined complexity of relational calculus is PSPACE-complete).

Some Interesting Datalog Programs

Example: Non 2-Colorability can be expressed by a Datalog program.

Fact: A graph E is 2-colorable if and only if it does not contain a cycle of odd length.

Datalog program for Non 2-Colorability:

```
ODD(x,y)    :--   E(x,y)
ODD(x,y)    :--   E(x,z), EVEN(z,y)
EVEN(x,y)   :--   E(x,z), ODD(z,y).
 Q          :--   ODD(x,x)
```

Sanity check: Can you find a Datalog program for Non 3-Colorability?

Some Interesting Datalog Programs

Example: Path Systems Problem

```
T(x) :--   A(x)
T(x) :--   R(x,y,z), T(y), T(z)
```

Theorem: S. Cook – 1974

Evaluating this Datalog program is a P-complete problem (via logspace-reductions).

Note:

- Path Systems was the first problem shown to be P-complete.
- In particular, it is highly unlikely that Path Systems is in NLOGSPACE or in LOGSPACE.
- In this sense, Datalog has higher data complexity than relational calculus.

Procedural Semantics of Datalog Programs

- On the face of it, the bottom-up evaluation of Datalog programs is an infinite process, since it gives rise to an infinite sequence of "stages" T^0, T^1, …, T^n, …
- Question: Does this infinite sequence converge?
- It will turn out that, for every Datalog program π and for every database instance I, the bottom-up evaluation of π on I terminates within a number of steps that is bounded by some polynomial in the size of I, i.e., there is some m such that
$$T^m = T^{m+1}.$$
Moreover, the smallest such m is bounded by some polynomial in the size of I.

Declarative vs. Procedural Datalog Semantics

Theorem: Let π be a Datalog program. Then the following are true:

- On every database instance I, the bottom-up evaluation of the procedural semantics of π terminates within a number of steps bounded by a polynomial in the size of I.
- The declarative semantics of π coincides with the procedural semantics of π.

Proof: For simplicity, assume that π has a single IDB T of arity k.

- By induction on n, show that $T^n \subseteq T^{n+1}$, for every n.

  (this uses the monotonicity of unions of conjunctive queries).
- Hence, $T^0 \subseteq T^1 \subseteq \ldots \subseteq T^n \subseteq T^{n+1} \subseteq \ldots$
- Since each $T^n \subseteq adom(I)^k$, there is an $m \leq |adom(I)|^k$ such that $T^m = T^{m+1}$.

Declarative vs. Procedural Datalog Semantics

Theorem: Let π be a Datalog program. Then the following are true:

- The bottom-up evaluation of the procedural semantics of π terminates within a number of steps bounded by a polynomial in the size of the database instance (= size of the EDB predicates).
- The declarative semantics of π coincides with the procedural semantics of π.

Proof: For simplicity, assume that π has a single IDB T of arity k.

- Since $T^m = T^{m+1}$, we have that the procedural semantics produces a solution to the recursive equation arising from π.
- By induction on n, show that if T* is another solution of this recursive equation, then $T^n \subseteq T^*$, for all n

  (use the monotonicity of unions of conjunctive queries again).
- In particular, $T^m \subseteq T^*$, hence T^m is the smallest solution of this recursive equation.

The Query Evaluation Problem for Datalog

Theorem: Let π be a Datalog program. There is a polynomial-time algorithm such that, given a database instance I, it evaluates π on I (i.e., it computes the semantics of π on I).

Proof: The bottom-up evaluation of the procedural semantics of π runs in polynomial time because:

- The number of iterations is bounded by a polynomial in the size of I.
- Each step of the iteration can be carried out in polynomial time (why?).

Corollary: The data complexity of Datalog is in P.

The Query Evaluation Problem for Datalog

Corollary: The data complexity of Datalog is in P.

Theorem: The combined complexity of Datalog is EXPTIME-complete.

Note:

- $P \subseteq NP \subseteq PSPACE \subseteq EXPTIME$.
- Moreover, it is known that P is properly contained in EXPTIME.
- Thus, Datalog has higher combined complexity than relational calculus (since the combined complexity of relational calculus is PSPACE-complete).

Some Interesting Datalog Programs

Example: Non 2-Colorability can be expressed by a Datalog program.

Fact: A graph E is 2-colorable if and only if it does not contain a cycle of odd length.

Datalog program for Non 2-Colorability:

```
ODD(x,y)    :--   E(x,y)
ODD(x,y)    :--   E(x,z), EVEN(z,y)
EVEN(x,y)   :--   E(x,z), ODD(z,y).
 Q          :--   ODD(x,x)
```

Sanity check: Can you find a Datalog program for Non 3-Colorability?

Some Interesting Datalog Programs

Example: Path Systems Problem

```
T(x) :-  A(x)
T(x) :-  R(x,y,z), T(y), T(z)
```

Theorem: S. Cook – 1974
Evaluating this Datalog program is a P-complete problem
(via logspace-reductions).

Note:

- Path Systems was the first problem shown to be P-complete.
- In particular, it is highly unlikely that Path Systems is in NLOGSPACE or in LOGSPACE.
- In this sense, Datalog has higher data complexity than relational calculus.

Linear Datalog

- Definition: A Datalog program is linear if the body of each rule contains at most one atomic formula involving an IDB predicate.

- Example: Linear Datalog Program for Transitive Closure

```
T(x,y) :- E(x,y)
T(x,y) :- E(x,z), T(z,y)
```

- Example: Non-linear Datalog program for Transitive Closure

```
T(x,y) :- E(x,y)
T(x,y) :- T(x,z), T(z,y)
```

Linear Datalog

Example: Give a linear Datalog program that computes the binary query COUSIN from the binary relation schema PARENT

```
SIBLING(x,y)  :-  PARENT(z,x), PARENT(z,y)
COUSIN(x,y)   :-  PARENT(z,x), PARENT(w,y), SIBLING(z,w)
COUSIN(x,y)   :-  PARENT(z,x), PARENT(w,y), COUSIN(z,w).
```

Fact: COUSIN(Barack Obama, Dick Chenney)
Actually, COUSIN^8(Barack Obama, Dick Chenney)

http://www.msnbc.msn.com/id/21340764/

Fact: COUSIN(Sarah Palin, Princess Diana).
Actually, COUSIN^{10}(Sarah Palin, Princess Diana)

http://www.dailymail.co.uk/news/worldnews/article-1073249/Sarah-Palin-Princess-Diana-cousins-genealogists-reveal.html

Linear Datalog

- Definition: A Datalog program π is linearizable if there is a linear Datalog program π^* that is equivalent to π.

- Example: The following Datalog program is linearizable:

```
T(x,y) :- E(x,y)
T(x,y) :- T(x,z), T(z,y)
```

- Example: The following Datalog program is **not** linearizable:

```
T(x) :-  A(x)
T(x) :-  R(x,y,z), T(y), T(z)
```

(the proof of this fact is non-trivial).

Datalog and SQL

- SQL:99 and subsequent versions of the SQL standard provide support for linear Datalog programs (but **not** for non-linear ones)

- Syntax:

WITH RECURSIVE T AS

<Datalog program for T>

<query involving T>

- Semantics:
  - Compute T as the semantics of <Datalog program for T>
  - The result of the previous step is a temporary relation that is then used, together with other EDBS, as if it were a stored relation (an EDB) in <query involving T>.

Datalog and SQL

Example: Give an SQL query for ANCESTOR (using PARENT as EDB)

```
 WITH RECURSIVE ANCESTOR(anc,desc)
(SELECT parent, child
 FROM    PARENT
 UNION
 SELECT PARENT.parent, ANCESTOR.desc
 FROM   PARENT, ANCESTOR
 WHERE PARENT.child = ANCESTOR.anc)
 SELECT * FROM ANCESTOR
```

Datalog and SQL

Example: Give an SQL query that computes all descendants of Noah

```
 WITH RECURSIVE ANCESTOR(anc,desc)
(SELECT parent, child
 FROM    PARENT
 UNION
 SELECT ANCESTOR.anc, PARENT.child
 FROM   PARENT, ANCESTOR
 WHERE PARENT.child = ANCESTOR.anc)
 SELECT desc FROM ANCESTOR
 WHERE anc = ‘Noah’
```

Datalog and SQL

- Linear Datalog programs with mutiple IDBs are supported in SQL:99
- Syntax:
  WITH RECURSIVE R, S, T, … AS
  <Datalog program for R, S, T, …>
  <query involving R, S, T, … >
- Semantics:
  - Compute R, S, T, … as the semantics of
    <Datalog program for R, S, T, …>
  - The result of the previous step are temporary relation that are then used, together with other EDBS, as if they were stored relation (EDBs) in <query involving R, S, T, …>.

Datalog(≠)

- Definition: Datalog(≠)
  - Datalog(≠) is the extension of Datalog in which the body of a rule may contain also ≠.
  - Declarative and Procedural Semantics of Datalog(≠) are similar to those of Datalog.

- Example: w-AVOIDING PATH

  Given a graph E and three nodes x,y, and w, is there a path from x to y that does not contain w?

  T(x,y,w) :- E(x,y), x ≠ w, y ≠ w

  T(x,y,w) :- E(x,z), T(z,y,w), x ≠ w.

Datalog with Negation

- Question: What if we allow negation in the bodies of Datalog rules?

- Examples:
  - T(x) :- ¬ T(x)
    (the recursive specification has **no** solutions!)
  - S(x) :- E(x,y), ¬ S(y)

- Note: Several different semantics for Datalog programs with negation have been proposed over the years (see Ch. 15 of AHV):
    - Stratified datalog programs
    - Well-founded semantics
    - Inflationary semantics
    - Stable model semantics
    - …

Query Equivalence and Containment for Datalog

- Note: Recall that the following are known about the Query Evaluation Problem for Datalog queries:
  - The data complexity of Datalog is in P.
  - The combined complexity of Datalog is EXPTIME-complete.

- Questions:
  - What about the Query Equivalence Problem for Datalog:
    Given two Datalog programs π and π', is π equivalent to π'?
    (do they return the same answer on every database instance?)
  - What about the Query Containment Problem for Datalog:
    Given two Datalog programs π and π', is $\pi \subseteq \pi'$?
    (is $\pi(I) \subseteq \pi'(I)$, on every database instance I?)

Query Equivalence and Containment for Datalog

Theorem: O. Shmueli – 1987

- The query equivalence problem for Datalog queries is undecidable. In fact, it is undecidable even for Datalog queries with a single IDB.
- Consequently, the query containment problem for Datalog queries is undecidable.

Hint of Proof:

- Reduction from Context-Free Grammar Equivalence:

  Given two context-free grammars G and G′, is L(G) = L(G′)?

  For more on this topic, read Chapter 12 of AHV.

The Complexity of Database Query Language

	Relational Calculus	Conjunctive Queries	Unions of Conjunctive Queries	Datalog Queries
Query Eval.: Combined Complexity	PSPACE-complete	NP-complete	NP-complete	EXPTIME-complete
Query Eval.: Data Complexity	In LOGSPACE (hence, in P)	In LOGSPACE (hence, in P)	In LOGSPACE (hence, in P)	P-complete
Query Equivalence	Undecidable	NP-complete	NP-complete	Undecidable
Query Containment	Undecidable	NP-complete	NP-complete	Undecidable

Composing Schema Mappings: An Overview

- This topic complements the topics on data integration and data exchange presented in the course by Maurizio Lenzerini.
- It is joint work with Ronald Fagin, Lucian Popa, and Wang-Chiew Tan.

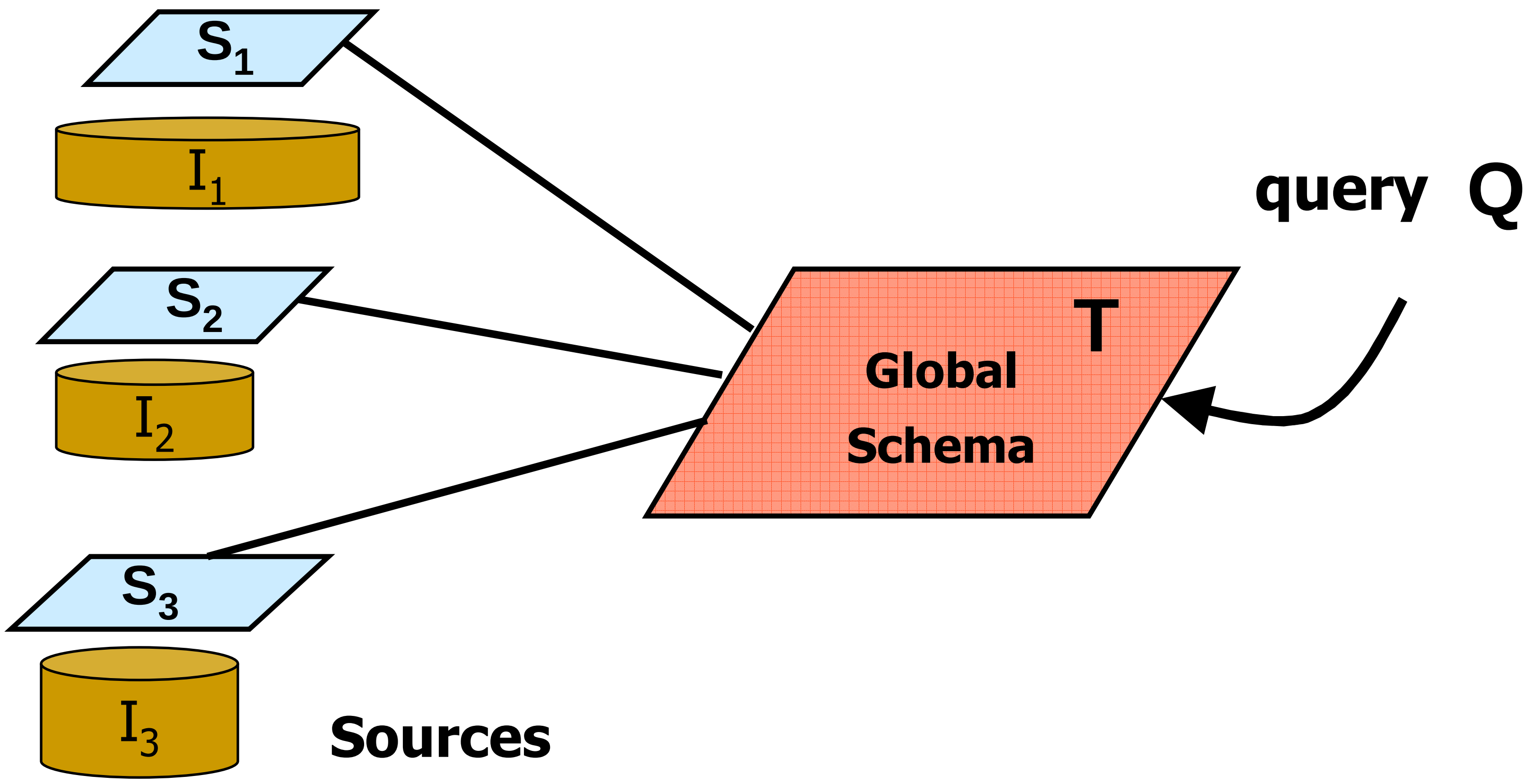
Theoretical Aspects of Data Interoperability

The research community has studied two different, but closely related, facets of data interoperability:

- **Data Integration** (aka **Data Federation**)
  - Formalized and studied for the past 10-15 years
- **Data Exchange** (aka **Data Translation**)
  - Formalized and studied for the past 5-6 years

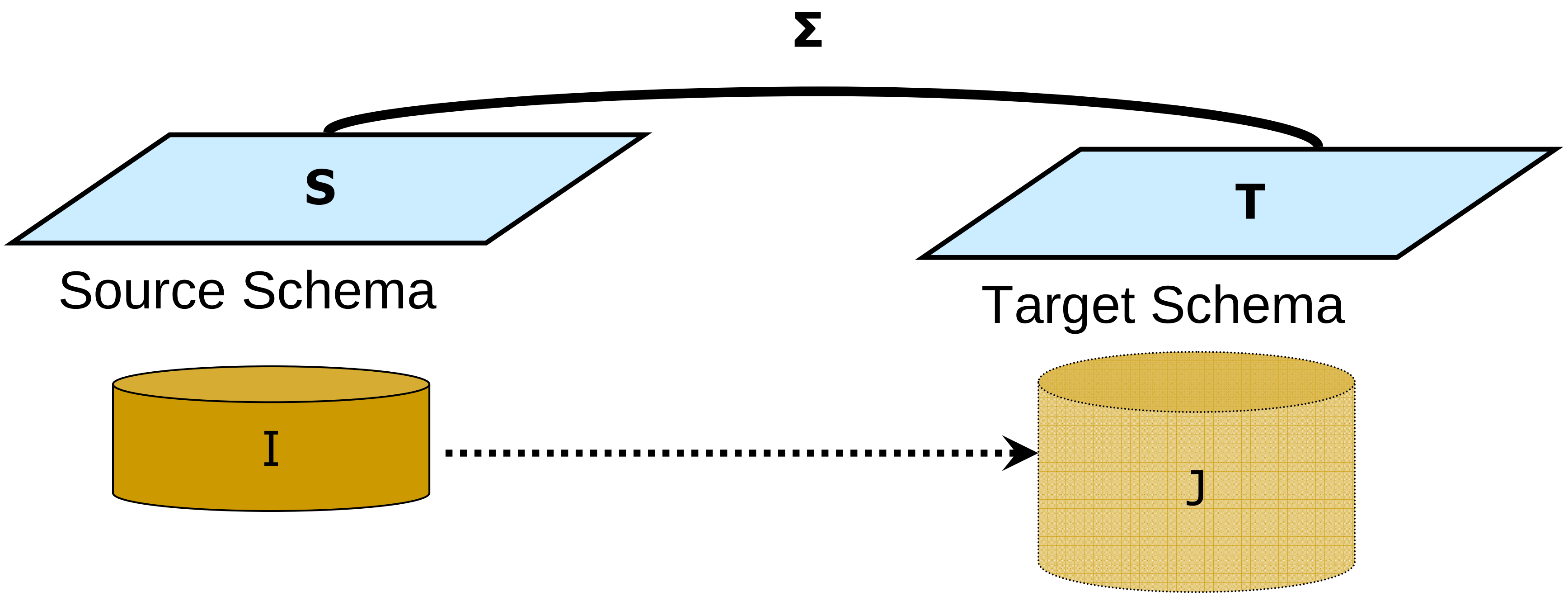
Data Integration

Query heterogeneous data in different sources via a virtual global schema



Data Exchange

Transform data structured under a source schema into data structured under a different target schema.



Schema Mappings

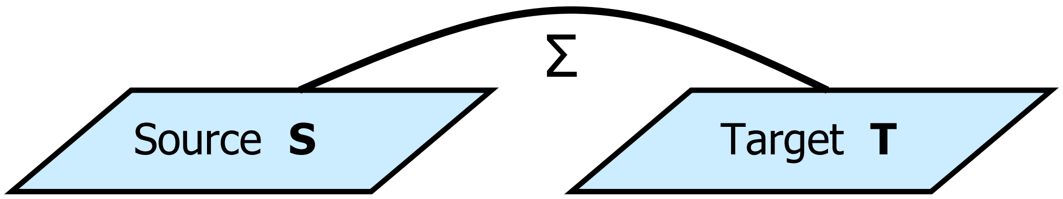
- Schema mappings:

  High-level, declarative assertions that specify the relationship between two database schemas.

- Schema mappings constitute the essential building blocks in formalizing and studying data interoperability tasks, including data integration and data exchange.

- Schema mappings make it possible to separate the design of the relationship between schemas from its implementation.
  - Are easier to generate and manage (semi)-automatically;
  - Can be compiled into SQL/XSLT scripts automatically.

Schema Mappings



- Schema Mapping **M** = (**S**, **T**, Σ)
  - Source schema **S**, Target schema **T**
  - High-level, declarative assertions Σ that specify the relationship between **S** and **T**.
- Question: What is a "good" schema-mapping specification language?

Schema-Mapping Specification Languages

- **Obvious Idea**:

  Use a logic-based language to specify schema mappings.

  In particular, use first-order logic.

- **Warning:**

  Unrestricted use of first-order logic as a schema-mapping specification language gives rise to **undecidability** of basic algorithmic problems about schema mappings.

Schema Mapping Specification Languages

Let us consider some simple tasks that every schema-mapping specification language should support:

- Copy (Nicknaming):
  - Copy each source table to a target table and rename it.
- Projection:
  - Form a target table by projecting on one or more columns of a source table.
- Column Augmentation:
  - Form a target table by adding one or more columns to a source table.
- Decomposition:
  - Decompose a source table into two or more target tables.
- Join:
  - Form a target table by joining two or more source tables.
- Combinations of the above (e.g., "join + column augmentation + …")

Schema Mapping Specification Languages

- Copy (Nicknaming):
  - $\forall x_1, \ldots, x_n (P(x_1,\ldots,x_n) \rightarrow R(x_1,\ldots,x_n))$
- Projection:
  - $\forall x,y,z (P(x,y,z) \rightarrow R(x,y))$
- Column Augmentation:
  - $\forall x,y\ (P(x,y) \rightarrow \exists z\ R(x,y,z))$
- Decomposition:
  - $\forall x,y,z\ (P(x,y,z) \rightarrow R(x,y) \wedge T(y,z))$
- Join:
  - $\forall x,y,z (E(x,z) \wedge F(z,y) \rightarrow R(x,y,z))$
- Combinations of the above (e.g., "join + column augmentation + …")
  - $\forall x,y,z (E(x,z) \wedge F(z,y) \rightarrow \exists w\ (R(x,y) \wedge T(x,y,z,w)))$

Schema Mapping Specification Languages

- Question: What do all these tasks (copy, projection, column augmentation, decomposition, join) have in common?

- Answer:
  - They can be specified using tuple-generating dependencies (tgds).

  - In fact, they can be specified using a special class of tuple-generating dependencies known as source-to-target tuple generating dependencies (s-t tgds).

Database Integrity Constraints

- Dependency Theory: extensive study of integrity constraints in relational databases in the 1970s and 1980s

  (Codd, Fagin, Beeri, Vardi …)

- Tuple-generating dependencies (tgds) emerged as an important class of constraints with a balance between high expressive power and good algorithmic properties. Tgds are expressions of the form

  $$\forall \mathbf{x} \, (\varphi(\mathbf{x}) \rightarrow \exists \mathbf{y} \, \psi(\mathbf{x}, \mathbf{y})),$$ where

  $\varphi(\mathbf{x})$, $\psi(\mathbf{x}, \mathbf{y})$ are conjunctions of atomic formulas.

  Special Cases:

  - Inclusion Dependencies
  - Multivalued Dependencies

Schema Mapping Specification Language

The relationship between source and target is given by source-to-target tuple generating dependencies (s-t tgds)

$\forall \mathbf{x}\ (\varphi(\mathbf{x}) \rightarrow \exists \mathbf{y}\ \psi(\mathbf{x}, \mathbf{y}))$, where

- $\varphi(\mathbf{x})$ is a conjunction of atoms over the source;
- $\psi(\mathbf{x}, \mathbf{y})$ is a conjunction of atoms over the target.

- s-t tgds assert that: some conjunctive query over the source is contained in some other conjunctive query over the target.

Example: (dropping the universal quantifiers in the front)

$(\text{Student}(s) \wedge \text{Enrolls}(s,c)) \rightarrow \exists t\ \exists g\ (\text{Teaches}(t,c) \wedge \text{Grade}(s,c,g))$

Schema Mapping Specification Language

Fact: s-t tgds generalize the main specifications used in data integration:

- They generalize LAV (local-as-view) specifications:

  $P(\mathbf{x}) \rightarrow \exists \mathbf{y}\, \psi(\mathbf{x}, \mathbf{y})$, where P is a source schema.

  - $E(x,y) \rightarrow \exists z\, (H(x,z) \wedge H(z,y))$ (LAV constraint)

Note: Copy, projection, and decomposition are LAV s-t tgds.

- They generalize GAV (global-as-view) specifications:

  $\varphi(\mathbf{x}) \rightarrow R(\mathbf{x})$, where R is a target relation

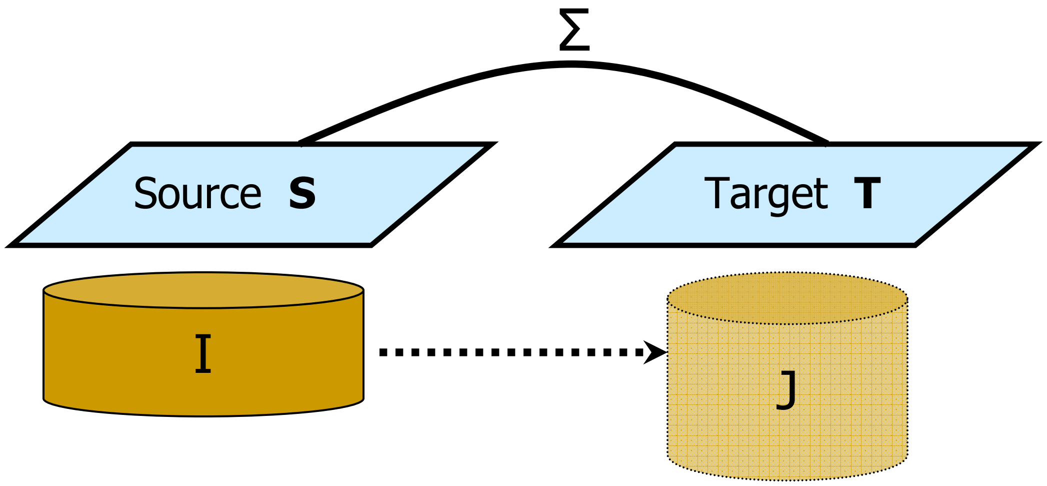
  (they are equivalent to full tgds: $\varphi(\mathbf{x}) \rightarrow \psi(\mathbf{x})$,

  where $\varphi(\mathbf{x})$ and $\psi(\mathbf{x})$ are conjunctions of atoms).

  - $E(x,y) \wedge E(y,z) \rightarrow F(x,z)$ (GAV (full) constraint)

Note: Copy, projection, and join are GAV s-t tgds.

Schema Mappings & Data Exchange



- Data Exchange via the schema mapping **M** = (**S**, **T**, Σ)

  Given a source instance I, construct a target instance J, so that (I, J) satisfy the specifications Σ of **M**.

  Such a J is called a solution for I.
- Difficulty:
  - Usually, there are multiple solutions
  - Which one is the "best" to materialize?

Data Exchange & Universal solutions

Fagin, K …, Miller, Popa:

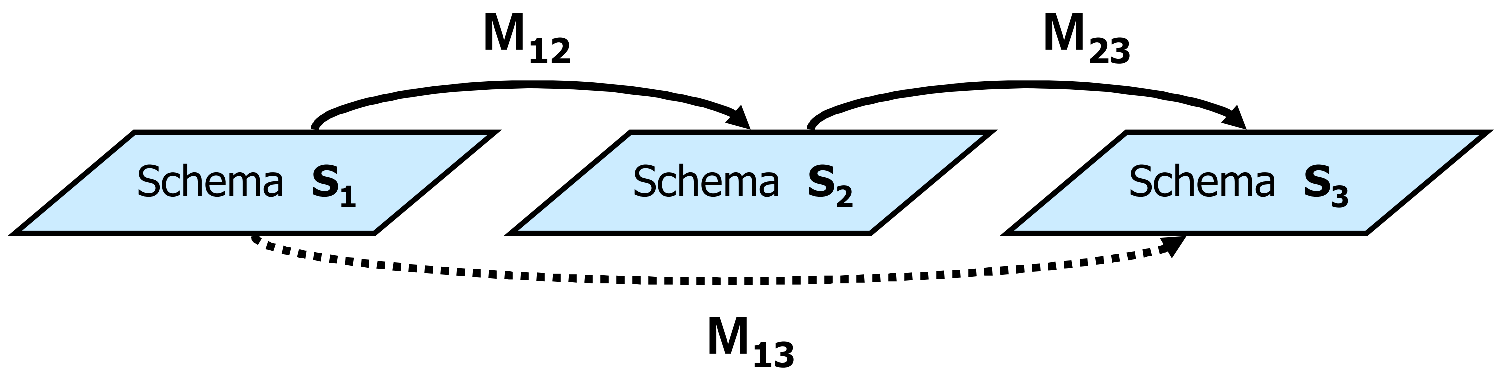
Identified and studied the concept of a universal solution for schema mappings specified by s-t tgds

- A universal solutions is a most general solution.
- A universal solution "represents" the entire space of solutions.
- A "canonical" universal solution can be generated efficiently using the chase procedure.
- A universal solution can be used to compute the certain answers of conjunctive queries over the target schema.

Managing Schema Mappings

- Schema mappings can be quite complex.
- Methods and tools are needed to automate or semi-automate schema-mapping management.
- Metadata Management Framework – Bernstein 2003
  based on generic schema-mapping operators:
  - Match operator
  - Merge operator
  - …
  - Composition operator
  - Inverse operator

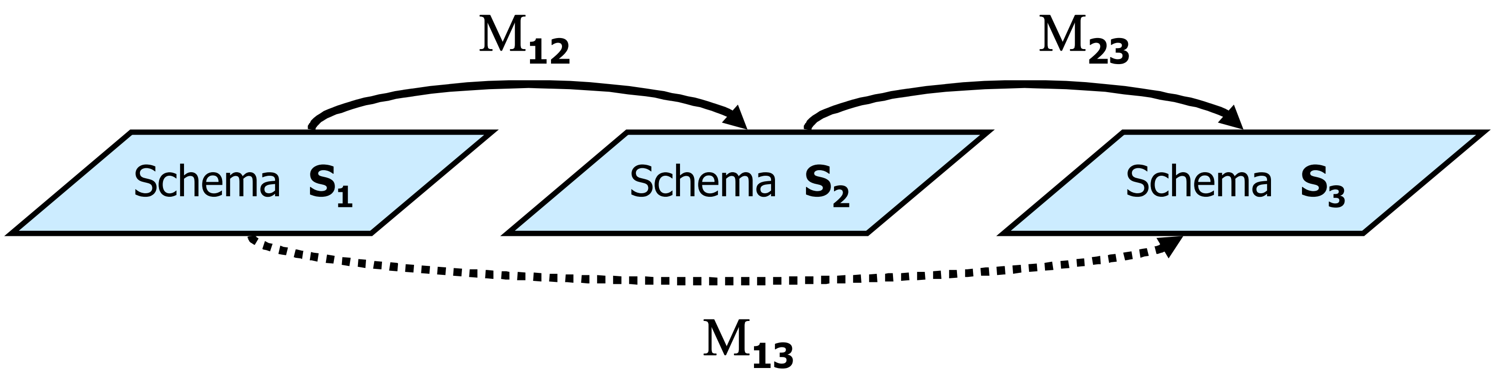
Composing Schema Mappings



- Given $\mathbf{M_{12}} = (\mathbf{S_1}, \mathbf{S_2}, \Sigma_{12})$ and $\mathbf{M_{23}} = (\mathbf{S_2}, \mathbf{S_3}, \Sigma_{23})$, derive a schema mapping $\mathbf{M_{13}} = (\mathbf{S_1}, \mathbf{S_3}, \Sigma_{13})$ that is "equivalent" to the sequential application of $\mathbf{M_{12}}$ and $\mathbf{M_{23}}$.
- $\mathbf{M_{13}}$ is a composition of $\mathbf{M_{12}}$ and $\mathbf{M_{23}}$

$$\mathbf{M_{13}} = \mathbf{M_{12}} \circ \mathbf{M_{23}}$$

Composing Schema Mappings



- Given $M_{12} = (\mathbf{S}_1, \mathbf{S}_2, \Sigma_{12})$ and $M_{23} = (\mathbf{S}_2, \mathbf{S}_3, \Sigma_{23})$, derive a schema mapping $M_{13} = (\mathbf{S}_1, \mathbf{S}_3, \Sigma_{13})$ that is "equivalent" to the sequence M_{12} and M_{23}.

What does it mean for M_{13} to be "equivalent" to the composition of M_{12} and M_{23}?

Earlier Work

- Metadata Model Management (Bernstein in CIDR 2003)
  - Composition is one of the fundamental operators
  - However, no precise semantics is given
- Composing Mappings among Data Sources
  (Madhavan & Halevy in VLDB 2003)
  - First to propose a semantics for composition
  - However, their definition is in terms of maintaining the same certain answers relative to a class of queries.
  - Their notion of composition *depends* on the class of queries; it may *not* be unique up to logical equivalence.

Semantics of Composition

- Every schema mapping **M** = (**S**, **T**, Σ) defines a binary relationship Inst(**M**) between instances*:*

  $$\mathrm{Inst}(\mathbf{M}) = \{ (I,J) \mid (I,J) \models \Sigma \}.$$

  In fact, from a semantic point of view, a schema mapping **M** can be identified with the set Inst(**M**).

- **Definition:** (FKPT)

  A schema mapping $\mathbf{M}_{13}$ is a composition of $\mathbf{M}_{12}$ and $\mathbf{M}_{23}$ if

  $$\mathrm{Inst}(\mathbf{M}_{13}) = \mathrm{Inst}(\mathbf{M}_{12}) \circ \mathrm{Inst}(\mathbf{M}_{23}),\ \text{that is,}$$

  $$(I_1,I_3) \models \Sigma_{13}$$

  if and only if

  there exists I_2 such that $(I_1,I_2) \models \Sigma_{12}$ and $(I_2,I_3) \models \Sigma_{23}$.

The Composition of Schema Mappings

Fact: If both $\mathrm{M} = (\mathbf{S}_1, \mathbf{S}_3, \Sigma)$ and $\mathrm{M}' = (\mathbf{S}_1, \mathbf{S}_3, \Sigma')$ are compositions of M_{12} and M_{23}, then Σ are Σ' are logically equivalent. For this reason:

- We say that M (or M'**)** is *the* composition of M_{12} and M_{23}.
- We write $\mathrm{M}_{12} \circ \mathrm{M}_{23}$ to denote it

Issues in Composition of Schema Mappings

- The semantics of composition was the first main issue.
- The second main issue is the language of the composition.
  - Is the language of s-t tgds *closed under composition*?

    If M_{12} and M_{23} are specified by finite sets of s-t tgds, is $M_{12} \circ M_{23}$ also specified by a finite set of s-t tgds?
  - If not, what is the "right" language for composing schema mappings?

Inexpressibility of Composition

Theorem:

- The language of s-t tgds is **not** closed under composition.

- In fact, there are schema mappings M_{12} and M_{23} specified by s-t tgds such that their composition $M_{12} \circ M_{23}$ is **not** expressible in least fixed-point logic LFP; hence, it is expressible neither in first-order logic nor in Datalog.

Lower Bounds for Composition

- M_{12} :
  $\forall x \forall y\, (E(x,y) \rightarrow \exists u \exists v\, (C(x,u) \wedge C(y,v)))$
  $\forall x \forall y\, (E(x,y) \rightarrow F(x,y))$
- M_{23} :
  $\forall x \forall y \forall u \forall v\, (C(x,u) \wedge C(y,v) \wedge F(x,y) \rightarrow D(u,v))$

- Given graph **G**=(V, E):
  - Let $I_1 = E$
  - Let $I_3 = \{ (r,g), (g,r), (b,r), (r,b), (g,b), (b,g) \}$

  Fact:
  G is 3-colorable iff $<I_1, I_3> \in \text{Inst}(M_{12}) \circ \text{Inst}(M_{23})$

- **Theorem (Dawar – 1998):**
  3-Colorability is **not** expressible in LFP

Complexity of Composition

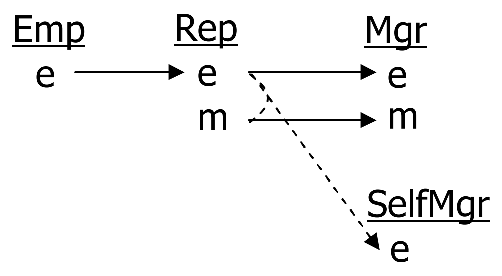
Definition: The model checking problem for a schema mapping **M** = (**S**, **T**, Σ) asks: given a source instanced I and a target instance J, does <I,J> $\models \Sigma$?

Fact: If a schema mapping M is specified by s-t tgds, then the model checking problem for M is in LOGSPACE.

Fact: There are schema mappings M_{12} and M_{23} specified by s-t tgds such that the model checking problem for their composition $M_{12} \circ M_{23}$ is NP-complete.

Employee Example

- Σ_{12} :
  - Emp(e) $\rightarrow$ $\exists$m Rep(e,m)
- Σ_{23} :
  - Rep(e,m) $\rightarrow$ Mgr(e,m)
  - Rep(e,e) $\rightarrow$ SelfMgr(e)



- **Theorem:** This composition is not definable by any set (finite or infinite) of s-t tgds.

- **Fact**: This composition is definable in a well-behaved fragment of second-order logic, called SO tgds, that extends s-t tgds with Skolem functions.

Employee Example - revisited

Σ_{12} :

- $\forall e\,(\ \mathrm{Emp}(e) \rightarrow \exists m\ \mathrm{Rep}(e,m)\ )$

Σ_{23} :

- $\forall e \forall m(\ \mathrm{Rep}(e,m) \rightarrow \mathrm{Mgr}(e,m)\ )$
- $\forall e\,(\ \mathrm{Rep}(e,e) \rightarrow \mathrm{SelfMgr}(e)\ )$

Fact: The composition is definable by the SO-tgd

Σ_{13} :

- $\exists \mathbf{f}\ (\forall e(\ \mathrm{Emp}(e) \rightarrow \mathrm{Mgr}(e,\mathbf{f(e)})\ ) \wedge$
  $\forall e(\ \mathrm{Emp}(e) \wedge (\mathbf{e=f(e)}) \rightarrow \mathrm{SelfMgr}(e)\ )\ )$

Second-Order Tgds

Definition: Let **S** be a source schema and **T** a target schema.

A second-order tuple-generating dependency (SO tgd) is a formula of the form:

$\exists f_1 \ldots \exists f_m((\forall \mathbf{x_1}(\phi_1 \rightarrow \psi_1)) \wedge \ldots \wedge (\forall \mathbf{x}_n(\phi_n \rightarrow \psi_n)))$, where

- Each f_i is a function symbol.
- Each ϕ_i is a conjunction of atoms from **S** and equalities of terms.
- Each ψ_i is a conjunction of atoms from **T.**

Example: $\exists \mathbf{f}\ (\forall e(\ Emp(e) \rightarrow Mgr(e,\mathbf{f(e)})\ ) \wedge$
$\forall e(\ Emp(e) \wedge (\mathbf{e=f(e)}) \rightarrow SelfMgr(e)\ )\ )$

Composing SO-Tgds and Data Exchange

Theorem (FKPT):

- The composition of two SO-tgds is definable by a SO-tgd.
- There is an algorithm for composing SO-tgds.
- The chase procedure can be extended to SO-tgds;
  it produces universal solutions in polynomial time.
- Every SO tgd is the composition of finitely many finite sets of s-t tgds. Hence, SO tgds are the "right" language for the composition of s-t tgds

When is the composition FO-definable?

Fact:

- It is an undecidable problem to tell whether the composition of two schema mappings specified by s-t tgds is first-order definable.
- However, there are certain sufficient conditions that guarantee that the composition is first-order definable.
  - If M_{12} is specified by GAV (full) s-t tgds and M_{23} is specified by s-t tgds, then their composition is definable by s-t tgds.
  - Arocena, Fuxman, Miller: If both M_{12} and M_{23} are specified by LAV s-t tgds with distinct variables, then their composition is specified by LAV s-t tgds.

Synopsis of Schema Mapping Composition

- s-t tgds are not closed under composition.
- SO-tgds form a well-behaved fragment of second-order logic.
  - SO-tgds are closed under composition; they are
    the "right" language for composing s-t tgds.
  - SO-tgds are "chasable":
    Polynomial-time data exchange with universal solutions.
- SO-tgds and the composition algorithm have been incorporated in Clio's Mapping Specification Language (MSL).

Related Work and Open Problems

Related Work:

- Composing richer schema mappings
  Nash, Bernstein, Melnik – 2007
- Composing Schema Mappings in Open & Closed Worlds
  Libkin and Sirangelo – 2008
- XML Schema Mappings
  Amano, Libkin, Murlak – 2009

Open Problems:

- Composition of schema mappings specified by s-t tgds and target constraints (target tgds and target egds).
- Composition of schema mappings specified by richer source-to-target dependencies.

“The notion of composition of maps leads to the most natural account of fundamental notions of mathematics, from multiplication, addition, and exponentiation, through the basic notions of logic."

"Conceptual Mathematics"
by
Lawevere and Schanuel