Logic and Databases

1st EATCS School for Young Researchers

Phokion G. Kolaitis
University of California Santa Cruz & IBM Research – Almaden

Logic and Databases

- Extensive interaction between logic and databases during the past 40 years.
- Logic provides both a unifying framework and a set of tools for formalizing and studying data management tasks.
- The interaction between logic and databases is a prime example of
  - Logic **in** Computer Science

    but also
  - Logic **from** Computer Science

Outline

Part I: Database Query languages: Expressive Power and Complexity

- Relational Algebra and Relational Calculus
- Conjunctive Queries and Homomorphisms
- Extensions of Conjunctive Queries with Unions and Inequalities
- Recursive Queries and Datalog

Unifying Theme:

- The use of logic as a database query language
- The interplay between databases, logic, and computational complexity

Outline

Part II: Database Dependencies and Data Exchange

- Functional and Inclusion Dependencies
- The Implication Problem for Database Dependencies
- Data-interoperability via Database Dependencies
- Schema Mappings and Data Exchange
- Managing Schema Mappings

Unifying Theme:

- The use of logic as a specification language in data management

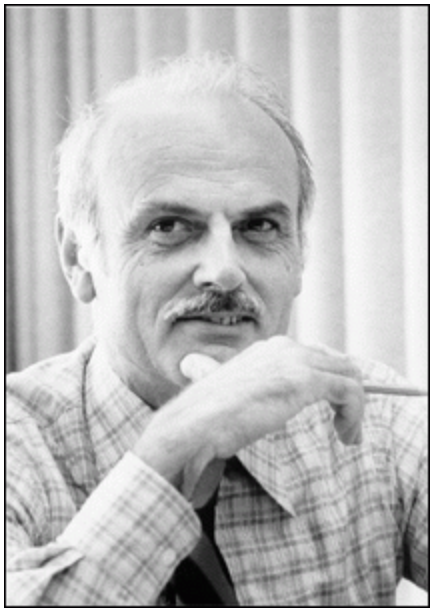
Relational Databases: How it all got started

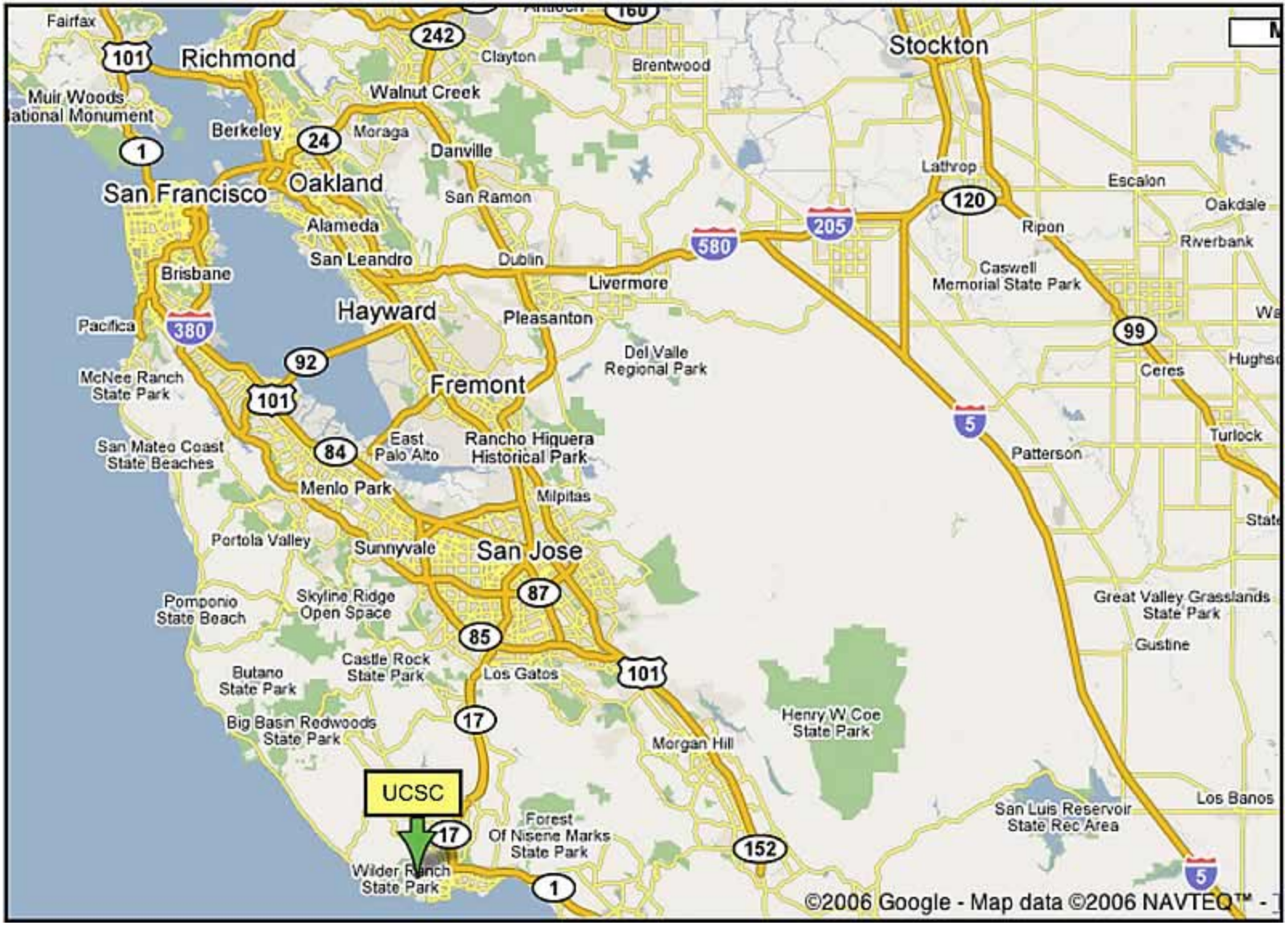
- The history of relational databases is the history of a scientific and technological revolution.

- 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




Fairfax
101
Richmond
242
Clayton
Brentwood
Stockton
Walnut Creek
Muir Woods
National Monument
Berkeley
24
Moraga
Danville
1
San Francisco
Oakland
San Ramon
Lathrop
Escalon
120
Oakdale
Alameda
205
Ripon
Riverbank
580
San Leandro
Dublin
Brisbane
Livermore
Caswell
Memorial State Park
Hayward
Pleasanton
Pacifica
380
99
92
Del Valle
Regional Park
Ceres
McNee Ranch
State Park
Fremont
101
5
Turlock
San Mateo Coast
State Beaches
84
East
Palo Alto
Rancho Higuera
Historical Park
Patterson
Menlo Park
Milpitas
Portola Valley
Sunnyvale
San Jose
Pomponio
State Beach
Skyline Ridge
Open Space
87
Great Valley Grasslands
State Park
85
Gustine
Butano
State Park
Castle Rock
State Park
Los Gatos
101
Henry W Coe
State Park
Big Basin Redwoods
State Park
17
Morgan Hill
UCSC
Los Banos
San Luis Reservoir
State Rec Area
Forest
Of Nisene Marks
State Park
17
152
Wilder Ranch
State Park
1
5
©2006 Google - Map data ©2006 NAVTEQ™

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 Table

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

Relation Schemas and Relational Database Schemas

- A k-ary relation schema **R**(A_1,A_2,...,A_k) is a set {A_1,A_2,...,A_k} of k attributes.
  - **CHECKING**(branch-name, account-no, customer-name, balance)
    - Thus, a k-ary relation schema is a "blueprint" for k-ary relations.
    - It is a k-ary relation symbol in logic with names for the positions.

- 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 $\mathbf{R_i}(A_1,A_2,...,A_{k_i})$, 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 Structures vs. Relational Databases

- Relational Structure

  $\mathbf{A} = (A, R_1,\ldots,R_m)$

  - A is the **universe** of **A**
  - $R_1,\ldots,R_m$ are the relations of **A**

- Relational Database

  $\mathbf{D} = (R_1,\ldots,R_m)$

- Thus, a relational database can be thought of as a relational structure **without** its universe.
  - And this causes some problems down the road ...

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?)

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.

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:

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

- Selection is a family of unary operations of the form

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

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 $\wedge$, $\vee$, $\neg$ applied to basic clauses.

- Examples:
  - balance < 1,000
  - branch-name = “Aptos”
  - σ (branch-name = “Aptos”) $\wedge$ (balance < 1,000) (SAVINGS)

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, ... | $(E_1 \cup E_2)$ | $(E_1 - E_2)$ | $(E_1 \times E_2)$ | $\pi_L(E)$ | $\sigma_\Theta(E)$,

  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).

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

- 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)

- 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 $(a_1,...,a_{r-s})$ such that for every tuple $(b_1,...,b_s)$ in S, we have that $(a_1,...,a_{r-s}, b_1,...,b_s)$ 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,

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

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.

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
WHERE Ψ

$$= \pi_{R_{i1}.A1, \ldots, R_{im}.Am} (\sigma_{\Psi} (R_1 \times \ldots \times R_K))$$

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.

Relational Calculus (First-Order Logic for Databases)

- First-order variables: x, y, z, ..., x_1, ...,x_k,...
  - They range over values that may occur in tables.
- Relation symbols: R, S, T, ... of specified arities (names of relations)
- Atomic (Basic) Formulas:
  - $R(x_1,...,x_k)$, where R is a k-ary relation symbol (alternatively, $(x_1,...,x_k) \in R$; the variables need not be distinct)
  - (x op y), where op is one of $=, \neq, <, >, \leq, \geq$
  - (x 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 as a Database Query Language

Definition:

- A relational calculus expression is an expression of the form

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

  where $\varphi(x_1,...,x_k)$ is a relational calculus formula with $x_1,...,x_k$ 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 $(a_1,...,a_k)$ 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.

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 ÷ S of two relations R and 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.

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

  Express R ÷ 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 ÷ 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.

The need for more formal semantics for Relational Calculus

- The semantics of the relational calculus expressions considered thus far have been unambiguous (and consistent with our intuition).
- However, consider the following relational calculus expressions:
  - $\{ (x_1,...,x_k): \neg R(x_1,...,x_k) \}$
  - $\{ x: \forall y,z \text{ ENROLLS}(x,y,z) \}$, with ENROLLS(s-name,course,term)
- **Question:** What is the semantics of each of these expressions?

The need for more formal semantics for Relational Calculus

Fact:

- To evaluate $\{ (x_1,...,x_k): \neg R(x_1,...,x_k) \}$ we need to know what the possible values for the variables x_1, ..., x_k are.
- If the variables $x_1,...,x_k$ range over a domain D, then

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

Note:

- Intuitively, the relational calculus expression

$$\{ (x_1,...,x_k): \neg R(x_1,...,x_k) \}$$

  is **not** "domain independent".
- In contrast, the relational calculus expression

  $\{ (x_1,...,x_k): S(x_1,..,x_k) \wedge \neg R(x_1,...,x_k) \}$ is "domain independent".

The need for more formal semantics for Relational Calculus

Note: The relational calculus expressions

- $\{ (x_1,...,x_k): \neg R(x_1,...,x_k) \}$

- $\{ x: \forall y,z \text{ ENROLLS}(x,y,z) \}$, with ENROLLS(s-name,course,term)

may produce different answers when we consider different domains over which the variables are interpreted.

Fact: Neither of these expressions is "domain independent".

The need for more formal semantics of Relational Calculus

- **Conclusion:**
  - To give rigorous and unambiguous semantics to relational calculus expressions, we need to make explicit the domain over which the variables (and the quantifiers) take values.
  - Is there a natural choice for the domain over which the variables take value?

Active Domain

Definition:

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

Example: Suppose that I consists of the relation R only, where

$$R = \{ (1,2,4), (5,61,4), (3,12,15), (17,2,7) \}$$

Then

$$adom(I) = \{ 1, 2, 4, 5, 61, 3, 12, 15, 17, 7 \}.$$

Active Domain and Relative Interpretations

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

$\varphi^{adom}(I)$ is the result of evaluating $\varphi(x_1,...,x_k)$ over adom(I) and I, that is, all variables and quantifiers are assumed to range over adom(I), and the relation symbols in φ are interpreted by the relations in I.

Example: Let φ be $\neg$ R(x,y)

- If R = {(1,2)}, then adom(R) = {1,2} and
  φ^{adom} (R) = {(2,1), (1,1), (2,2)}

- If R = {(1,2),(3,1)}, then adom(R) = {1,2,3} and
  φ^{adom} (R) = {(2,1), (1,1), (2,2), (1,3), (2,3), (3,2), (3,3)}

Active Domain and Relative Interpretation

Example: Let φ be $\forall y R(x,y)$

- If R = {(1,2), (1,1)}, then
  - adom(φ) = {1,2}
  - φ^{adom} (R) = {1}.
- If R = {(1,2), (1,1), (3,1)}, then
  - adom(φ) = {1,2,3}
  - $\varphi^{adom}(R) = \emptyset$

  (1 is not in $\varphi^{adom}(R)$, because (1,3) is not in R).

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 relational calculus formula φ such that q(I) = φ^{adom} (I), for every database instance I
   (in other words, q is expressible in relational calculus under the active domain interpretation).

From Relational Algebra to Relational Calculus

Theorem: For every relational expression E, there is an equivalent relational calculus expression $\{ (x_1,...,x_k): \varphi(x_1,...x_k) \}$, i.e., $E(I) = \varphi^{adom}(I)$

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

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

From Relational Algebra to Relational Calculus

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

Proof: (continued)

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

From Relational Algebra to Relational Calculus

Example: R(A,B), S(C,D)

Translate $\pi_{1,4}(\sigma_{R.B=S.C}(R \times S))$ to relational calculus

1. R translates to R(x,y)
2. S translates to S(z,w)
3. $R \times S$ translates to $R(x,y) \wedge S(z,w)$
4. $\sigma_{R.B=S.C}(R \times S)$ translates to $(y=z) \wedge R(x,y) \wedge S(z,w)$
5. $\pi_{1,4}(\sigma_{R.B=S.C}(R \times S))$ translates to
   $\exists y \exists z ((y=z) \wedge R(x,y) \wedge S(z,w))$
   or, simply, to $\exists y (R(x,y) \wedge S(y,w))$.

So, we have the rel. calculus expression $\{(x,w): \exists y (R(x,y) \wedge S(y,w))\}$.

From Relational Calculus to Relational Algebra

Proof (Sketch):

2. $\Rightarrow$ 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 induction on the construction of relational calculus formulas and the above fact 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 relational calculus under the active domain interpretation, and
  - an algorithm from translating from relational calculus (under the active domain interpretation) to relational algebra.

- Each of these two algorithms runs in linear time in the size of the input (relational algebra expression or relational calculus formula).

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 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 $q \equiv q \wedge q'$.

Undecidability

Theorem: (Turing - 1936)

The Halting Problem is undecidable.

- Given a Turing Machine M and an input x, does M halt on x?

Theorem: (Trakhtenbrot – 1949):

The Finite Validity Problem is undecidable:

- Given a first-order sentence ψ, is ψ true on all finite graphs?

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

    φ 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.

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.

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 φ 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 $I \models \varphi$?

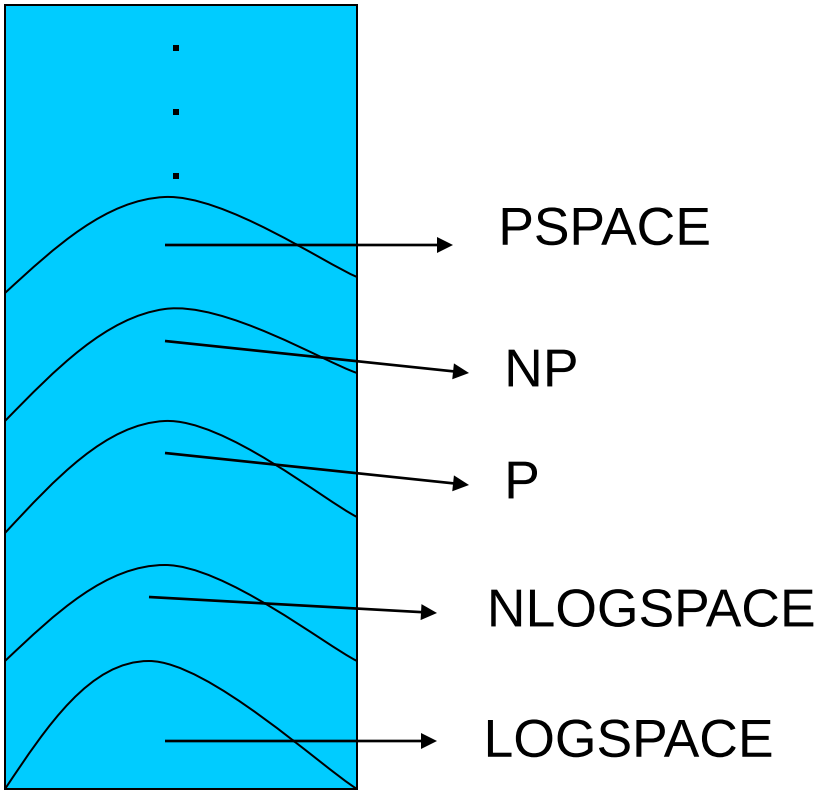
Vardi's Taxonomy of the Query Evaluation Problem

- 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

Summary

- Relational Algebra and Domain Independent Relational Calculus have the same expressive power.
- The Query Equivalence Problem for Relational Calculus in undecidable.
- The Query Containment Problem for Relational Calculus is undecidable.
- The Query Evaluation Problem for Relational Calculus is PSPACE-complete.

Three Fundamental Algorithmic Problems (reminder)

- 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
  $q(I) = q'(I)$?)
  - Logical equivalence is a special case

- 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
  $q(I) \subseteq q'(I)$?)
  - Logical implication is a special case

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.

  $$\{(x_1,\ldots,x_k): \exists z_1 \ldots \exists z_m \chi(x_1, \ldots,x_k, z_1,\ldots,z_k)\},$$

  where $\chi(x_1, \ldots,x_k, z_1,\ldots,z_k)$ 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 $R(y_1,...,y_n)$, and $\wedge$ and $\exists$ only.

$$\{(x_1,...,x_k): \exists z_1 ... \exists z_m \chi(x_1, ...,x_k, z_1,...,z_k)\}$$

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

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

    - 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.

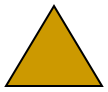
- Question: What is the common link?
- Answer: The Homomorphism Problem

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 $(a_1,...,a_m)$, we have that
  if $(a_1,...,a_m) \in P^I$, 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



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

  $R_i(a_1,...,a_m)$ such that $(a_1,...,a_m) \in R_i$.

- 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^{K_3} of K_3 is given by the rule:

  Q^{K_3} :- E(x,y),E(y,x),E(x,z),E(z,x),E(y,z),E(z,y)

- The canonical conjunctive query Q^{K_3} 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 $P^{I^Q}=\{z\}$

- Basic Fact:

  For every conjunctive query Q, we have that $I^Q \models Q$.

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,...,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,..., 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. $\Rightarrow$ 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. $\Rightarrow$ 3. It follows from the other direction of the Magic Lemma.

3. $\Rightarrow$ 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 : $\exists x_1 \exists x_2 \, (E(x_1,x_2) \wedge E(x_2,x_1))$
  - 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: $G \rightarrow K_3$
- $K_3 \models Q^G$
- $Q^{K3} \subseteq Q^G$.

The Homomorphism Theorem for non-Boolean Conjunctive Queries

The Homomorphism Theorem: Chandra & Merlin – 1977

Consider two k-ary conjunctive queries

$Q(x_1,...,x_k)$:-- $R_1(\mathbf{u}_1)$, ..., $R_n(\mathbf{u}_n)$ and $Q'(x_1,...,x_k)$:-- $T_1(\mathbf{v}_1)$, ..., $T_m(\mathbf{v}_m)$,

Then the following are equivalent:

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

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 $Q^{K3} \subseteq Q^{G}$.

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 $Q^H \equiv Q^{K3}$.

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

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.

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

1. $q_1 \cup q_2 \cup ... \cup q_m \subseteq q'_1 \cup q'_2 \cup ... \cup q'_n$.
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 $I^{q_i} \models q_1 \cup q_2 \cup ... \cup q_m$, hence $I^{q_i} \models q'_1 \cup q'_2 \cup ... \cup q'_n$, 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 $I^{q'_{k_i}}$ 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

Limitations of Relational Algebra & Relational Calculus

Outline:

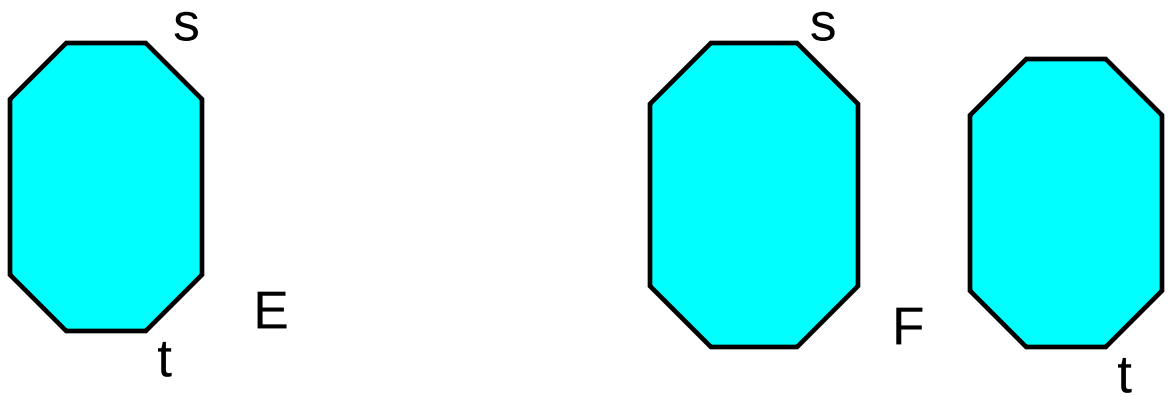
- Relational Algebra and Relational Calculus have substantial expressive power. In particular, they **can** express
  - Natural Join
  - Quotient
  - Unions of conjunctive queries
  - ...
- However, they **cannot** express recursive queries.
- Datalog is a declarative database query language that augments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

Transitive Closure and Relational Calculus

Theorem: A. Aho and J. Ullman – 1979 (really, Fraïssé – 1954)
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.

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:

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

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

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).

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:

$$T^0 = \varnothing$$

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

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

$$\begin{aligned} ODD(x,y) &:- E(x,y) \\ ODD(x,y) &:- E(x,z), EVEN(z,y) \\ EVEN(x,y) &:- E(x,z), ODD(z,y) \end{aligned}$$

- Bottom-up evaluation:

$$\begin{aligned} ODD^0 &= \varnothing \\ EVEN^0 &= \varnothing \\ ODD^{n+1} &= \{(a,b): E(a,b) \vee \exists\, z(E(a,z) \wedge EVEN^n(z,b))\} \\ EVEN^{n+1} &= \{(a,b): \exists\, z(E(a,z) \wedge ODD^n(z,b))\} \end{aligned}$$

Fact: The following statements are true:

- $\cup_{n \geq 1} ODD^n = \{ (a,b)$: there is a path of odd length from a to b $\}$
- $\cup_{n \geq 1} EVEN^n = \{ (a,b)$: 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 $T^n \subseteq \text{adom}(I)^k$, there is an $m \leq |\text{adom}(I)|^k$ 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

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 Cheney)
Actually, $COUSIN^8$(Barack Obama, Dick Cheney)

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 vs. First-Order Logic

Facts:

- Unions of Conjunctive Qureries **are contained** in Datalog
- Relational Calculus **is not contained** in Datalog
  - The quotient operation is not expressible in Datalog
- Datalog **is not contained** in Relational Calculus
  - Transitive closure is not expressible in Relational Calculus

Corollary to Rossman's Theorem:

Datalog ∩ Relational Calculus = Unions of Conjunctive Queries

Proof:

Datalog queries are preserved under homomorphisms.

Datalog vs. First-Order Logic

Theorem (Ajtai-Guvevich – 1987)

The following statements are equivalent for a Datalog program π:

1. π is bounded.
2. The query q defined by π is expressible in first-order logic.

Proof: Use Rossman's Theorem (2005) for: (2) implies (1).

- If q is first-order expressible, then, by Rossman, q is equivalent to a finite union of conjunctive queries $q'_1 \cup ... \cup q'_m$.
- Therefore,

$$q_1 \cup ... \cup q_n \cup ... \equiv q'_1 \cup ... \cup q'_m$$

- By Sagiv and Yannakakis, it follows that there is some N such that

$$q_1 \cup ... \cup q_n \cup ... \equiv q_1 \cup ... \cup q_N.$$

- Hence, π is bounded.

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')?

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