CMPS 277 – Principles of Database Systems

https://courses.soe.ucsc.edu/courses/cmps277/Fall11/01

Lecture #5
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 $\phi$ such that $q(I) = \varphi^\text{adom} (I)$, for every database instance $I$ (in other words, $q$ is expressible in domain independent relational calculus).

3. There is a relational calculus formula $\psi$ such that $q(I) = \psi^\text{adom} (I)$, 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

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 \( R(x,y) \land S(z,w) \)
4. \( \sigma_{R.B=S.C} (R \times S) \) translates to \( (y=z) \land R(x,y) \land S(z,w) \)
5. \( \pi_{1,4} (\sigma_{R.B=S.C} (R \times S)) \) translates to
   \[ \exists y \exists z ((y=z) \land R(x,y) \land S(z,w)) \]
   or, simply, to
   \[ \exists y (R(x,y) \land S(y,w)) \]
Equivalence of Relational Algebra and Relational Calculus

Proof (Sketch):
1. ⇒ 2.
   - We also need to show that the resulting formula is domain independent.
   - Show by induction that this translation of relational algebra to relational calculus is actually a translation of relational algebra to domain independent relational calculus.

2. ⇒ 3. This implication is obvious.

3. ⇒ 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 $\text{adom}(I) = E(I)$.

   - 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.
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 $\forall y R(x,y)$.
  - $\forall y R(x,y) \equiv \neg \exists y \neg R(x,y)$
  - $\text{dom}(I) = \pi_1(R) \cup \pi_2(R)$

<table>
<thead>
<tr>
<th>Rel.Calc. formula $\varphi$</th>
<th>Relational Algebra Expression for $\varphi^{\text{dom}}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\neg R(x,y)$</td>
<td>$(\pi_1(R) \cup \pi_2(R)) \times (\pi_1(R) \cup \pi_2(R)) - R$</td>
</tr>
<tr>
<td>$\exists y \neg R(x,y)$</td>
<td>$\pi_1((\pi_1(R) \cup \pi_2(R)) \times (\pi_1(R) \cup \pi_2(R)) - R)$</td>
</tr>
<tr>
<td>$\neg \exists y \neg R(x,y)$</td>
<td>$(\pi_1(R) \cup \pi_2(R)) - (\pi_1((\pi_1(R) \cup \pi_2(R)) \times (\pi_1(R) \cup \pi_2(R)) - R))$</td>
</tr>
</tbody>
</table>
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.
Roadmap

Thus far, we have:

- Introduced and studied the foundations of the relational data model.
- Established the equivalence between relational algebra and relational calculus as regards to the expressibility of queries.
- Illustrated the influence of relational algebra and relational calculus on SQL (for more on SQL, see the slides on “The Influence of Relational Algebra and Relational Calculus on SQL” at the course webpages).

The next step will be to:

- Introduce certain fundamental problems about database queries.
- Study the algorithmic properties of these problems for queries expressible in relational algebra/relational calculus.
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 $\text{dom}(I)$.

- 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 $q(I) = 1$.

**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?
- Given a graph $E$ (binary relation), is $E$ 3-colorable?
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 $q(I) \subseteq q'(I)$?)
Examples of Query Equivalence

Let R and S be binary relational schemas. Then

- $\pi_1(R \cup S)$ is equivalent to $\pi_1(R) \cup \pi_1(S)$

- $\pi_1(R \cap S)$ is not equivalent to $\pi_1(R) \cap \pi_1(S)$

- $\{ x: \forall y(R(x,y) \land S(x,y)) \}$ is equivalent to $\{ x: \forall yR(x,y) \land \forall yS(x,y) \}$

- $\{ x: \forall y(R(x,y) \lor S(x,y)) \}$ is not equivalent to $\{ x: \forall yR(x,y) \lor \forall yS(x,y) \}$. 
Examples of Query Containment

Let $R$ and $S$ be binary relational schemas. Then

- $\pi_1(R \cap S)$ is contained in $\pi_1(R) \cap \pi_1(S)$

- $\pi_1(R) \cap \pi_1(S)$ is **not** contained in $\pi_1(R \cap S)$

- $\{ x: \forall y R(x,y) \lor \forall y S(x,y) \}$ is contained in $\{ x: \forall y (R(x,y) \lor S(x,y)) \}$

- $\{ x: \forall y (R(x,y) \lor S(x,y)) \}$ is **not** contained in $\{ x: \forall y R(x,y) \lor \forall y S(x,y) \}$. 
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 the **Query Equivalence Problem** are *closely related* to each other:
  1. \( q \equiv q' \) if and only if \( q \subseteq q' \) and \( q' \subseteq q \).
  2. \( q \subseteq q' \) if and only if \((q \land q') \equiv q\). (Why?)
Our goal is to investigate the algorithmic aspects of these problems for queries expressible in relational algebra/relational calculus (i.e., the queries are given via relational algebra/relational calculus expressions that define them)

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, NP, PSPACE, and complete problems for NP and for PSPACE.
Definition (informal): A decision problem $Q$ consists of a set of inputs and a question with a “yes” or “no” answer for each input.

<table>
<thead>
<tr>
<th>Input $x$</th>
<th>$Q$?</th>
<th>$1$ (“yes”)</th>
<th>$0$ (“no”)</th>
</tr>
</thead>
</table>

Definition:
- $Σ^*$ is the set of all strings over a finite alphabet $Σ$.
- A language over $Σ$ is a set $L \subseteq Σ^*$.
- Every language $L$ gives rise to the following decision problem:
  - Given $x \in Σ^*$, 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 \( f : \Sigma^* \rightarrow \Sigma^* \)

- Church’s Thesis (aka Church-Turing Thesis): The following statements are equivalent for a (partial) function \( f : \Sigma^* \rightarrow \Sigma^* \):
  - 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 $\chi$ 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 $x \in L$
    - $s_L(x) = \text{undefined}$ 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 sentence $\varphi$ on graphs, is $\varphi$ 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

- **The 3n +1 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 → 34 → 17 → 52 → 26 → 13 → 40 → 20 → 10 
    → 5 → 16 → 8 → 4 → 2 → 1.

- **Open Problem:** Does this program terminate on every input?
  - Raised for the first time by Lothar Collatz in 1937 - see http://en.wikipedia.org/wiki/Collatz_conjecture
Undecidable Problems

- **The Finite Validity Problem (B. Trakhtenbrot – 1949):** Given a first-order sentence $\varphi$ on graphs, is $\varphi$ true on every finite graph?
  
- **Examples of Finitely Valid Formulas:**
  - $\forall x (E(x,x) \rightarrow \exists y E(x,y))$
  - $\forall x \forall y (E(x,x) \land x = y \rightarrow E(y,y))$
  - “if $E$ is a total order, then $E$ has a biggest element”

- **Example of Non-Finitely Valid Sentences:**
  - $\forall x \forall y (E(x,y) \rightarrow E(y,x))$
  - $(\forall x \exists y E(x,y)) \rightarrow (\exists y \forall x E(x,y))$

- 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 $p(x_1,\ldots,x_n)$ with integer coefficients, does $p(x_1,\ldots,x_n)$ have an all-integers root? (i.e., does the equation $p(x_1,\ldots,x_n) = 0$ 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

$$x \in L \iff f(x) \in L^*.$$ 

- Such a function $f$ is called a reduction of $L$ to $L^*$.
- $L \preceq L^*$ 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 \(\leq\) Finite Validity Problem.

- Many database problems have been shown to be undecidable via reductions from one of the following problems:
  - The Halting Problem
  - The Finite Validity Problem
  - Hilbert’s 10\(^{th}\) Problem.

- In particular, Di Paola proved that the Domain Independence Problem for relational calculus formulas is undecidable by showing that Finite Validity Problem \(\leq\) Domain Independence.
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 $\preceq$ Query Equivalence Problem
- To see, let $\psi^*$ be a fixed finitely valid relational calculus sentence (say, $\forall x (E(x,x) \rightarrow \exists y E(x,y))$).
- Then, for every relational calculus sentence $\varphi$, we have that $\varphi$ is finitely valid $\iff \varphi \equiv \psi^*$. 
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 $\preceq$ Query Containment, since
  
  $q \equiv q' \iff q \subseteq q'$ and $q' \subseteq q$.

- **Notice the chain of reductions:**
  
  Halting Problem $\preceq$ Finite Validity $\preceq$ Query Equiv. $\preceq$ 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.