CMPS 277 – Principles of Database Systems

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

Lecture #12
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.
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.
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
  - Static program analysis (trace recursive calls)

SQL:1999 and subsequent versions of the SQL standard provide support for a sublanguage of Datalog, called linear Datalog.
Definition: A Datalog program $\pi$ is a finite set of rules each expressing a conjunctive query

$$T(x_1,\ldots,x_k) :-- R_1(u_1), \ldots, R_n(u_n),$$

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)
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

- 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 \( \pi, \sigma \), 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:**
  
  \[
  T(x,y) \leftarrow E(x,y)
  \]
  
  \[
  T(x,y) \leftarrow T(x,z), T(z,y)
  \]

- Recursive equation:
  
  \[
  T = E \cup \pi_{1,4}(\sigma_{2=3}(T \times T))
  \]
Declarative Semantics of Datalog Programs

Example: Consider the Datalog program:

\[
\begin{align*}
\text{ODD}(x,y) & :\text{-} \ E(x,y) \\
\text{ODD}(x,y) & :\text{-} \ E(x,z), \ \text{EVEN}(z,y) \\
\text{EVEN}(x,y) & :\text{-} \ E(x,z), \ \text{ODD}(z,y).
\end{align*}
\]

- System of recursive equations:

\[
\begin{align*}
\text{ODD} & = E \cup \pi_{1,4}(\sigma_{2=3}(E \times \text{EVEN})) \\
\text{EVEN} & = \pi_{1,4}(\sigma_{2=3}(E \times \text{ODD})).
\end{align*}
\]
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 with a single IDB has a smallest solution (smallest w.r.t. the \( \subseteq \) partial order).

- **Example:** Datalog program
  
  \[
  \begin{align*}
  T(x,y) & : - E(x,y) \\
  T(x,y) & : - T(x,z), T(z,y)
  \end{align*}
  \]

  - 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 with a single IDB 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?
Example: Consider again the Datalog program:

\[
\begin{align*}
\text{ODD}(x,y) & :\neg\text{E}(x,y) \\
\text{ODD}(x,y) & :\neg\text{E}(x,z), \text{EVEN}(z,y) \\
\text{EVEN}(x,y) & :\neg\text{E}(x,z), \text{ODD}(z,y).
\end{align*}
\]

- **System** of recursive equations:
  \[
  \begin{align*}
  \text{ODD} & = \text{E} \cup \pi_{1,4}(\sigma_{2=3}(\text{E} \times \text{EVEN})) \\
  \text{EVEN} & = \pi_{1,4}(\sigma_{2=3}(\text{E} \times \text{ODD})).
  \end{align*}
  \]

  The **smallest solution** of this system is a pair of relations \((P,Q)\) such that
  1. \((P,Q)\) satisfies this system (e.g., \(Q = \pi_{1,4}(\sigma_{2=3}(\text{E} \times P))\)).
  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.
Definition: Let $\pi$ be a Datalog program. The procedural semantics of $\pi$ are obtained by the following **bottom-up evaluation** of the recursive predicates (IDBs) of $\pi$:

1. Set all IDBs of $\pi$ to $\emptyset$.
2. Apply all rules of $\pi$ 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.
Example: Datalog program for Transitive Closure

\[
\begin{align*}
T(x,y) & : - \ E(x,y) \\
T(x,y) & : - \ E(x,z), T(z,y)
\end{align*}
\]

- **Bottom-up evaluation:**
  \[
  T^0 = \emptyset \\
  T^{n+1} = \{(a,b): E(a,b) \lor \exists z(E(a,z) \land T^n(z,b))\}
  \]

**Fact:** The following statements are true:

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

**Proof:** By induction on \( n. \)
Theorem: Let \( \pi \) be a Datalog program. Then the following are true:

- The bottom-up evaluation of the procedural semantics of \( \pi \) 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 \( \pi \) coincides with the procedural semantics of \( \pi \).

Proof: For simplicity, assume that \( \pi \) 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, \( T^0 \subseteq T^1 \subseteq ... \subseteq T^n \subseteq T^{n+1} \subseteq ... \)
- 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 $\pi$ be a Datalog program. Then the following are true:

- The bottom-up evaluation of the procedural semantics of $\pi$ 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 $\pi$ coincides with the procedural semantics of $\pi$.

**Proof:** For simplicity, assume that $\pi$ 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 $\pi$.
- 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 $\pi$ be a Datalog program. There is a polynomial-time algorithm such that, given a database instance $I$, it evaluates $\pi$ on $I$ (i.e., it computes the semantics of $\pi$ on $I$).

**Proof:** The bottom-up evaluation of the procedural semantics of $\pi$ 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:

$$
\begin{align*}
\text{ODD}(x,y) & :\quad \text{E}(x,y) \\
\text{ODD}(x,y) & :\quad \text{E}(x,z), \text{EVEN}(z,y) \\
\text{EVEN}(x,y) & :\quad \text{E}(x,z), \text{ODD}(z,y). \\
\text{Q} & :\quad \text{ODD}(x,x)
\end{align*}
$$

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.
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

\[
\begin{align*}
T(x,y) & : - E(x,y) \\
T(x,y) & : - E(x,z), T(z,y)
\end{align*}
\]

Example: Non-linear Datalog program for Transitive Closure

\[
\begin{align*}
T(x,y) & : - E(x,y) \\
T(x,y) & : - T(x,z), T(z,y)
\end{align*}
\]
Linear Datalog

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

\[
\begin{align*}
\text{SIBLING}(x,y) & : - \text{PARENT}(z,x), \text{PARENT}(z,y) \\
\text{COUSIN}(x,y) & : - \text{PARENT}(z,x), \text{PARENT}(w,y), \text{SIBLING}(z,w) \\
\text{COUSIN}(x,y) & : - \text{PARENT}(z,x), \text{PARENT}(w,y), \text{COUSIN}(z,w).
\end{align*}
\]

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


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

Linear Datalog

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

- **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).