CMPS 277 – Principles of Database Systems

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

Lecture #13
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.
Example: Consider again the Datalog program:

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

- **System** of recursive equations:

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

- The **smallest solution** of this system is a pair of relations \((P,Q)\) such that
  1. \((P,Q)\) satisfies this system (e.g., \(Q = \pi_{1,4}(\sigma_{2=3}(E \times P))\)).
  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.
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, $T^0 \subseteq T^1 \subseteq \ldots \subseteq T^n \subseteq T^{n+1} \subseteq \ldots$
- 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
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: Path Systems Problem

\[ T(x) :\leftarrow A(x) \]
\[ T(x) :\leftarrow R(x,y,z), T(y), T(z) \]

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

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

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

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

- **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)

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

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

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:

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

Example: The following Datalog program is not linearizable:

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

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

WITH RECURSIVE ANCESTOR(anc,desc) 
(SELECT parent, child 
FROM   PARENT 
UNION 
SELECT PARENT.parent, ANCESTOR.desc 
FROM   PARENT, ANCESTOR 
WHERE PARENT.child = ANCESTOR.anc) 
SELECT * FROM ANCESTOR
Example: Give an SQL query that computes all descendants of Noah

```
WITH RECURSIVE ANCESTOR(anc, desc)
(SELECT parent, child
FROM PARENT
UNION
SELECT ANCESTOR.anc, PARENT.child
FROM PARENT, ANCESTOR
WHERE PARENT.child = ANCESTOR.anc)
SELECT desc FROM ANCESTOR
WHERE anc = 'Noah'
```
Linear Datalog programs with multiple IDBs are supported in SQL:99

Syntax:
WITH RECURSIVE R, S, T, ... AS
<Datalog program for R, S, T, ...>
<query involving R, S, T, ... >

Semantics:
- Compute R, S, T, ... as the semantics of
  <Datalog program for R, S, T, ...>
- The result of the previous step are temporary relation that are then used, together with other EDBS, as if they were stored relation (EDBs) in <query involving R, S, T, ...>. 
**Datalog(≠)**

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

- **Example:** *w-AVOIDING PATH*
  Given a graph E and three nodes x, y, and w, is there a path from x to y that does not contain w?
  
  \[
  T(x,y,w) : \neg E(x,y), \ x \neq w, \ y \neq w \\
  T(x,y,w) : \neg E(x,z), \ T(z,y,w), \ x \neq w.
  \]
Datalog with Negation

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

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

- **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 $\pi$ and $\pi'$, is $\pi$ equivalent to $\pi'$? (do they return the same answer on every database instance?)
  - What about the **Query Containment Problem** for Datalog: Given two Datalog programs $\pi$ and $\pi'$, is $\pi \subseteq \pi'$? (is $\pi(I) \subseteq \pi'(I)$, on every database instance $I$?)
The query equivalence problem for Datalog queries is undecidable. In fact, it is undecidable even for Datalog queries with a single IDB. Consequently, the query containment problem for Datalog queries is undecidable.

Hint of Proof:

- Reduction from Context-Free Grammar Equivalence:
  Given two context-free grammars $G$ and $G'$, is $L(G) = L(G')$?

For more on this topic, read Chapter 12 of AHV.
# The Complexity of Database Query Language

<table>
<thead>
<tr>
<th></th>
<th>Relational Calculus</th>
<th>Conjunctive Queries</th>
<th>Unions of Conjunctive Queries</th>
<th>Datalog Queries</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Query Eval.: Combined Complexity</strong></td>
<td>PSPACE-complete</td>
<td>NP-complete</td>
<td>NP-complete</td>
<td>EXPTIME-complete</td>
</tr>
<tr>
<td><strong>Query Eval.: Data Complexity</strong></td>
<td>In LOGSPACE (hence, in P)</td>
<td>In LOGSPACE (hence, in P)</td>
<td>In LOGSPACE (hence, in P)</td>
<td>In P; It can be P-complete</td>
</tr>
<tr>
<td><strong>Query Equivalence</strong></td>
<td>Undecidable</td>
<td>NP-complete</td>
<td>NP-complete</td>
<td>Undecidable</td>
</tr>
<tr>
<td><strong>Query Containment</strong></td>
<td>Undecidable</td>
<td>NP-complete</td>
<td>NP-complete</td>
<td>Undecidable</td>
</tr>
</tbody>
</table>
Database Dependencies

- Codd introduced *functional dependencies* in 1972.
- Soon after this, several different classes of integrity constraints were introduced and studied:
  - Multivalued Dependencies
  - Join Dependencies
  - Inclusion Dependencies
  - ....

- Eventually, it was realized that:
  - Most of the integrity constraints considered are special cases of *embedded implicational dependencies*.
  - *Embedded implicational dependencies* can be expressed by formulas of *relational calculus*. 
Database Dependencies

Outline:

- Will introduce functional dependencies (FDs) and related notions (superkeys, candidate keys, keys).
- Will study the Implication Problem for functional dependencies.
- Will introduce the class of embedded implicational dependencies.
- Will introduce other classes of database dependencies as special cases of embedded implicational dependencies.
**Functional Dependencies**

**Definition:** Let $R$ be a relational schema and $r$ an instance of $R$.

- If $A_1, \ldots, A_m, B$ are attributes of $R$, then we say that $r$ satisfies the functional dependency
  \[ A_1, \ldots, A_m \rightarrow B \]
  if whenever two tuples in $r$ agree on the values of $A_1, \ldots, A_m$, then they also agree on the value of $B$.
  (in other words, there are no two tuples in $r$ that have the same value on the attributes of $A_1, \ldots, A_m$, but differ on the value of $B$).

- If $A_1, \ldots, A_m, B_1, \ldots, B_k$ are attributes of $R$, then we say that $r$ satisfies the functional dependency
  \[ A_1, \ldots, A_m \rightarrow B_1, \ldots, B_k \]
  if $r$ satisfies the functional dependencies
  \[ A_1, \ldots, A_m \rightarrow B_1, \]
  \[ \ldots \]
  \[ A_1, \ldots, A_m \rightarrow B_k. \]
Functional Dependencies

Example: \( R(A,B,C,D,E) \)

- \( r \) satisfies:

- \( r \) does not satisfy:

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>3</td>
<td>4</td>
<td>6</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>4</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>5</td>
<td>1</td>
<td>2</td>
</tr>
</tbody>
</table>
Functional Dependencies

Example: \( R(A,B,C,D,E) \)

- \( r \) satisfies:
  - \( A,B \rightarrow D \)
  - \( C \rightarrow D \)
  - \( B \rightarrow A \)
  - \( A,B,C \rightarrow D,E \)
  - ...

- \( r \) does not satisfy:
  - \( A \rightarrow B \)
  - \( C \rightarrow B \)
  - \( D \rightarrow E \)
  - ...

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>3</td>
<td>4</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>4</td>
<td>4</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>5</td>
<td>1</td>
<td>2</td>
<td></td>
</tr>
</tbody>
</table>
Functional Dependencies

- **Definition:** A relational schema $R$ satisfies the functional dependency
  \[ A_1, \ldots, A_m \rightarrow B_1, \ldots, B_k \]
  if every instance $r$ of $R$ satisfies $A_1, \ldots, A_m \rightarrow B_1, \ldots, B_k$.

- **Fact:** In effect, the above definition imposes a semantic restriction
  on the instances of $R$, namely, we disallow all instances that violate
  the functional dependency $A_1, \ldots, A_m \rightarrow B_1, \ldots, B_k$.

- **Notation:** If $X$ and $Y$ are sets of attributes of $R$, then we write
  \[ X \rightarrow Y \]
  to denote the functional dependency with the members of $X$ in the
  left-hand side and the members of $Y$ in the right-hand side.
Functional Dependencies

Question: How do we know that a FD holds for a database schema?

Answer:
- This is semantic information that is provided by the customer who wishes to have a database schema designed for the data of interest.
- A FD may be derived (inferred) from other known FDs about the schema (much more on this later).