Database Theory: Beyond FO

CS 645
Feb 26, 2008

Some slide content based on materials of Dan Suciu, Ullman/Widom
Coming lectures

• TODAY:
  – Limited expressiveness of FO
  – Adding recursion (Datalog)
  – Expressiveness & Complexity
Query language classes

FO queries

Recursive Queries

Conjunctive Queries

Expressiveness

Algebra

Logic

SQL

RA

(safe) RC

UCQ−

CQ<

CQ≠

CQ−

RA:

σ,π,×

single datalog rule

SdFW

UNION

EXCEPT

SFW +
Review

• CQs are an important fragment of FO
  – Equivalences: RA: $\sigma,\pi,\times$  SQL: $S^dFW$
  – Properties: satisfiable, monotonic
  – containment/equivalence decidable, NPc

• Expressiveness
  – CQs strictly less expressive than FO

• Hardness of static optimization
Expressive Power of FO

• Let $I = \{R(x,y)\}$ represent a graph
• Query $\text{path}(x,y) =$
  – all $x, y$ such that there is path from $x$ to $y$

• Theorem: $\text{path}(x,y)$ cannot be expressed in FO.
Datalog Programs

- A **Datalog program** is a collection of rules.
- In a program, subgoals can be either
  1. EDB = **Extensional Database** = stored table.
  2. IDB = **Intensional Database** = computed table.
- Never both! No EDB in heads.
Non-recursive rules

Graph:  \( R(x,y) \)

\[
P(x,y) :\ - R(x,u), \ R(u,v), \ R(v,y) \\
A(x,y) :\ - P(x,u), \ P(u,y)
\]

Can “unfold” it into:

\[
A(x,y) :\ - R(x,u), \ R(u,v), \ R(v,w), \ R(w,m), \ R(m,n), \ R(n,y)
\]
Example: Datalog Program

• Using EDB \texttt{Sells(bar, beer, price)} and \texttt{Beers(name, manf)}, find the manufacturers of beers Joe doesn’t sell.

\begin{align*}
\text{JoeSells(b)} & \leftarrow \text{Sells(’Joes Bar’, b, p)} \\
\text{Answer(m)} & \leftarrow \text{Beers(b,m)} \land \neg\text{JoeSells(b)}
\end{align*}
Evaluating Datalog Programs

• As long as there is no recursion, we can pick an order to evaluate the IDB predicates, so that all the predicates in the body of its rules have already been evaluated.
• If an IDB predicate has more than one rule, each rule contributes tuples to its relation.
Expressive Power of Datalog

- Without recursion, Datalog expresses exactly the first order queries
  - negated subgoals
  - implicit union

- We call this
  - “non-recursive Datalog with negation”
  - denoted: nr-Datalog
Query language classes

- FO queries
- Recursive Queries

Expressiveness

- Conjunctive Queries

RA:
\( \sigma, \pi, \chi \)

- Conjunctive Queries
- Recursive Queries

RA:
(safe) RC
UCQ

nr-Datalog\(^-\)

SFW +
UNION
EXCEPT

UCQ
CQ\(<\)
CQ\(\neq\)
CQ\(^-\)
Recursive example

Two forms of transitive closure:

Graph:  \( R(x,y) \)

Path\((x,y)\) :- \( R(x,y) \)
Path\((x,y)\) :- Path\((x,u)\), \( R(u,y) \)

Path\((x,y)\) :- \( R(x,y) \)
Path\((x,y)\) :- Path\((x,u)\), Path\((u,y)\)
Recursive Example

- EDB: Par(c,p) = p is a parent of c.
- Generalized cousins: people with common ancestors one or more generations back:
  \[
  \text{Sib}(x,y) :- \text{Par}(x,p), \text{Par}(y,p), x \neq y \\
  \text{Cousin}(x,y) :- \text{Sib}(x,y) \\
  \text{Cousin}(x,y) :- \text{Par}(x,x_p), \text{Par}(y,y_p), \text{Cousin}(x_p,y_p)
  \]
Definition of Recursion

• Form a **dependency graph** whose nodes = IDB predicates.

• Arc X -> Y if and only if there is a rule with X in the head and Y in the body.

• Cycle = recursion; no cycle = no recursion.
Example: Dependency Graphs

Recursive

Sib(x, y) :- Par(x, p), Par(y, p), x ≠ y
Cousin(x, y) :- Sib(x, y)

Nonrecursive

Cousin(x, y) :- Par(x, x_p), Par(y, y_p), Cousin(x_p, y_p)
## Variants of Datalog

<table>
<thead>
<tr>
<th></th>
<th>without recursion</th>
<th>with recursion</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>without ¬</strong></td>
<td>Non-recursive Datalog = UCQ</td>
<td>Datalog</td>
</tr>
<tr>
<td><strong>with ¬</strong></td>
<td>Non-recursive Datalog$^\neg$ = FO</td>
<td>Datalog$^\neg$</td>
</tr>
</tbody>
</table>
Query language classes

Recursive Queries

FO queries

Conjunctive Queries

Expressiveness

Algebra

Logic

SQL

RA

Datalog

Datalog

nr-Datalog

(safe) RC

UCQ

UCQ

CQ<

CQ≠

CQ¬

RA: σ,π,×

single datalog rule

SFW +

UNION

EXCEPT

SdFW
Meaning of Datalog rules

• Model theoretic
  – datalog rules define a set of satisfying relations
    • whenever body true, head is true

• Proof theoretic
  – Set of fact derivable from EDB relations by applying the rules
Evaluating Recursive Rules

- The following works when there is no negation:
  1. Start by assuming all IDB relations are empty.
  2. Repeatedly evaluate the rules using the EDB and the previous IDB, to get a new IDB.
  3. End when no change to IDB.

“Fixed Point”
The “Naïve” Evaluation Algorithm

Start:
IDB = ∅

Apply rules to IDB, EDB

Change to IDB?

yes

no
done
Example: Evaluation of Cousin

• We’ll proceed in rounds to infer Sib facts and Cousin facts.
• Remember the rules:

\[
\text{Sib}(x,y) \leftarrow \text{Par}(x,p) \land \text{Par}(y,p) \land x \neq y \\
\text{Cousin}(x,y) \leftarrow \text{Sib}(x,y) \\
\text{Cousin}(x,y) \leftarrow \text{Par}(x,xp) \land \text{Par}(y,yp) \\
& \quad \text{Cousin}(xp,yp)
\]
Seminaive Evaluation

• Since the EDB never changes, on each round we only get new IDB tuples if we use at least one IDB tuple that was obtained on the previous round.

• Saves work; lets us avoid rediscovering most known facts.
  - A fact could still be derived in a second way.
Par Data: Parent Above Child

Sibling
Cousin

Round 1
Round 2
Round 3
Round 4
Datalog (no ¬)

- There are three equivalent meanings for a datalog rule
  - least fixed point
  - (unique) minimal model
  - set of facts derivable from EDBs
Recursion Plus Negation

• “Naïve” evaluation doesn’t work when there are negated subgoals.

• In fact, negation wrapped in a recursion makes no sense in general.

• Even when recursion and negation are separate, we can have ambiguity about the correct IDB relations.
Stratified Negation

• Stratification is a constraint usually placed on Datalog with recursion and negation.
• It rules out negation wrapped inside recursion.
• Gives the sensible IDB relations when negation and recursion are separate.
Another example

\[ P(x) :: R(x) \land \neg Q(x) \]
\[ Q(x) :: R(x) \land \neg P(x) \]

- Suppose \( R = \{(1)\} \)
- Two models that satisfy rules:
  - \( P = \{\} \quad Q = \{1\} \)
  - \( P = \{1\} \quad Q = \{\} \)
- Neither is minimal
Strata

• Intuitively, the **stratum** of an IDB predicate $P$ is the maximum number of negations that can be applied to an IDB predicate used in evaluating $P$.

• Stratified negation = “finite strata.”
Stratum Graph

• To formalize strata use the stratum graph:
  - Nodes = IDB predicates.
  - Arc A -> B if predicate A depends on B.
  - Label this arc “−” if the B subgoal is negated.
Stratified Negation Definition

• The **stratum** of a node (predicate) is the maximum number of – arcs on a path leading from that node.

• A Datalog program is **stratified** if all its IDB predicates have finite strata.
Another example

\[ P(x) :- R(x) \& \neg Q(x) \]
\[ Q(x) :- R(x) \& \neg P(x) \]

- Not stratified
Another Example

• EDB = \text{Source}(x), \text{Target}(x), \text{Arc}(x,y).
• Rules for "targets not reached from any source":

  Reach(x) \leftarrow \text{Source}(x)
  \quad \text{Reach}(x) \leftarrow \text{Reach}(y) \text{ AND } \text{Arc}(y,x)
  \quad \text{NoReach}(x) \leftarrow \text{Target}(x) \& \neg \text{Reach}(x)
The Stratum Graph

Stratum 1:
\( \leq 1 \) arc on any path out.

Stratum 0:
No – arcs on any path out.
The Stratified Model

• When the Datalog program is stratified, we can evaluate IDB predicates lowest-stratum-first.
• Once evaluated, treat it as EDB for higher strata.
Assumption

• When the logic is stratified, the stratified model is the one that “makes sense.”
• This principle is used in SQL-99 recursion --- the stratified model is defined to be the correct query result.
Variants of Datalog

<table>
<thead>
<tr>
<th></th>
<th>without recursion</th>
<th>with recursion</th>
</tr>
</thead>
<tbody>
<tr>
<td>without ¬</td>
<td>Non-recursive Datalog = UCQ</td>
<td>Datalog</td>
</tr>
<tr>
<td>with ¬</td>
<td>Non-recursive Datalog¬ = FO</td>
<td>Datalog¬</td>
</tr>
</tbody>
</table>
Query language classes

- **Recursive Queries**
  - FO queries
  - RA: $\sigma, \pi, \times$
  - (safe) RC
  - UCQ
  - $\neg$UCQ

- **Conjunctive Queries**
  - nr-Datalog
  - Datalog

- **Logic**
  - SFW +
  - UNION
  - EXCEPT

- **Algebra**
  - SQL
Containment for Datalog

• **Theorem** Containment for datalog programs is undecidable.
Overview of complexity of containment

<table>
<thead>
<tr>
<th></th>
<th>Complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>CQ</td>
<td>NP-complete</td>
</tr>
<tr>
<td>UCQ</td>
<td>NP-complete</td>
</tr>
<tr>
<td>nr-Datalog (no ¬)</td>
<td>Π^p_2</td>
</tr>
<tr>
<td>CQ¬</td>
<td>Π^p_2</td>
</tr>
<tr>
<td>CQ&lt;</td>
<td>Π^p_2</td>
</tr>
<tr>
<td>FO (nr-Datalog)</td>
<td>undecidable</td>
</tr>
<tr>
<td>Datalog</td>
<td>undecidable</td>
</tr>
</tbody>
</table>
Query evaluation complexity

- **Data complexity**: the complexity of evaluating a fixed query for variable database inputs

- **Query complexity**: the complexity of evaluating, on a fixed db instance, the queries specifiable in a given query language.
FO queries

- The data complexity for the (safe) FO queries is LOGSPACE.

- The basic operations of relational queries are efficient to compute.
Descriptive complexity