Datalog

Nicoleta Preda

(based on the book Abiteboul & al: Foundations of Databases)

December 6, 2017

Example: Database Instance

Links	*Line*	*Station*	*Next Station*
	4	*St.-Germain*	*Odeon*
	4	*Odeon*	*St.-Michel*
	4	*St.-Michel*	*Chatelet*
	1	*Chatelet*	*Louvre*
	1	*Louvre*	*Palais-Royal*
	1	*Palais-Royal*	*Tuileries*
	1	*Tuileries*	*Concorde*
	9	*Pont de Sevres*	*Billancourt*
	9	*Billancourt*	*Michel-Ange*
	9	*Michel-Ange*	*Iena*
	9	*Iena*	*F. D. Roosevelt*
	9	*F. D. Roosevelt*	*Republique*
	9	*Republique*	*Voltaire*

Example: Queries

Q_1: Can we go from Odeon to Chatelet?

Q_2: What are the stations reachable from Odeon?

Q_3: What lines can be reached from Odeon?

Example: Queries

Q_1: Can we go from Odeon to Chatelet?

Q_2: What are the stations reachable from Odeon?

Q_3: What lines can be reached from Odeon?

- None of the above queries can be expressed in SQL!
- We need a query language where we can express recursion.

Conjunctive Queries

Definition: Rule (Rule-Based Conjunctive Query)

Let $\mathcal{R}$ be a database schema. A rule-based conjunctive query over $\mathcal{R}$ is an expression of the form

$$q(u) \leftarrow R_1(u_1), ..., R_n(u_n)$$

where

- $n \geq 0$,
- q is a relation name not in $\mathcal{R}$;
- $R_1, \ldots, R_n$ are relation names in $\mathcal{R}$;
- $u, u_1, \ldots, u_n$ are free tuples (i.e., may use either variables or constants)
- and each variable occurring in u must also occur at least once in $u, u_1, \ldots, u_n$.

Semantics

Let q be the query given earlier, *var(q)* be the set of variables in the head, and *dom* be the set of constants and let $I(R)$ be an instance of R.

Query Answer

The image of I under q is $q(I) = \{v(u) | v$ is a valuation[1] over *var(q)* and $v(u_i) \in I(R_i))$, for each $i \in [1, n]\}$.

[1] A valuation is a function.

Equivalence Theorem

The rule-based conjunctive queries and satisfiable SPJR algebra are equivalent.

SPJR algebra: algebra with 4 operators:

- Selection,
- Projection,
- (natural) Join,
- Renaming

Incorporating Union: Non-Recursive Datalog

A nonrecursive datalog program over schema $\mathcal{R}$ is a set of rules where

- no relation name in $\mathcal{R}$ occurs in a rule head;
- the same relation name may appear in more than one rule head;
- and there is some ordering $r_1, ..., r_m$ of the rules so that the relation name in the head of r_i does not occur in the body of a rule r_j whenever $j \leq i$.

Introducing Recursive Rules

Example:

$\text{ancestor}(x, z) \leftarrow \text{parent}(x, y), \text{ancestor}(y, z)$

Exercise:

Define the transitive closure for the graph:

Links	*Line*	*Station*	*Next Station*
	4	*St.-Germain*	*Odeon*
	4	*Odeon*	*St.-Michel*
	4	*St.-Michel*	*Chatelet*
	1	*Chatelet*	*Louvre*
	1	*Louvre*	*Palais-Royal*
	1	*Palais-Royal*	*Tuileries*
	1	*Tuileries*	*Concorde*
	9	*Pont de Sevres*	*Billancourt*

Datalog (with recursion)

Definition: Datalog Rule

A (datalog) rule is an expression of the form

$$R_1(u_1) \leftarrow R_2(u_2), ..., R_n(u_n)$$

where

- $R_1, \ldots, R_n$ are relation names;
- $u_1, \ldots, u_n$ are free tuples (i.e., may use either variables or constants)
- and each variable occurring in u_1 must also occur at least once in $u_2, \ldots, u_n$.

A datalog program is a finite set of datalog rules.

Extensional *versus* Intensional

Let P be a datalog program.

- An *extensional* relation is a relation occurring only in the body of the rules.
- An *intensional* relation is a relation occurring in the head of some rule of P.

- $edb(P)$: the extensional (database) schema, which consists of the set of all extensional relation names;
- $idb(P)$: the intensional schema, which consists of all the intensional relation names;
- $sch(P) = edb(P) \cup idb(P)$ is the schema of P.

Datalog Semantics

The semantics of a datalog program is a mapping from database instances over *edb(P)* to database instances over *idb(P)*.

We call the input data the extensional database and the program the intensional database.

Exercise

Given an extensional database schema consisting of the relation name

Links(Line, Station, NextStation)

propose a Datalog Program that can answer the following three queries:

Q_1: Can we go from Odeon to Chatelet?

Q_2: What are the stations reachable from Odeon?

Q_3: What lines can be reached from Odeon?

Datalog *versus* Logic Programming

Logic programming permits function symbols, but datalog does not.

Evaluation: Fix-Point Solution

Let P be a datalog program and K an instance over *sch(P)*.

A fact A is an immediate consequence for K and P if

- either $A \in K(R)$ for some *edb* relation R,
- or $A \leftarrow A_1, \ldots, A_n$ is an instantiation of a rule in P and each A_i is in K.

The immediate consequence operator of P, denoted T_P, is the mapping from *inst(sch(P))* to *inst(sch(P))* defined as follows. For each K, $TP(K)$ consists of all facts A that are immediate consequences for K and P.

- The operator T_P is monotone.
- K is a fix point of T if $T_P(K) = K$
- For each P and instance I, T_P has a minimum fixpoint containing I.

Example: Datalog program with negation and no fix point

Consider the following program in Datalog^{-}

$$R(x) \leftarrow S(x), \neg R(x) \tag{1}$$

and suppose that the EDB S has input $\{S(1)\}$. In this case, neither $\{R(1)\}$ nor $\{\neg R(1)\}$ are fixpoints! In fact, the above program does not have any fixpoint if we follow the standard definition.

Evaluation: Proof-Theoretic Solution

Idea: The answer of a program P on I consists of the set of facts that can be proven using P and I.

A proof tree of a fact A from I and P is a labeled tree where

1. each vertex of the tree is labeled by a fact;
2. each leaf is labeled by a fact in I;
3. the root is labeled by A; and
4. for each internal vertex, there exists an instantiation $A_1 \leftarrow A_2, \ldots, A_n$ of a rule in P such that the vertex is labeled A_1 and its children are respectively labeled $A_2, \ldots, A_n$.

Such a tree provides a proof of the fact A.

Example

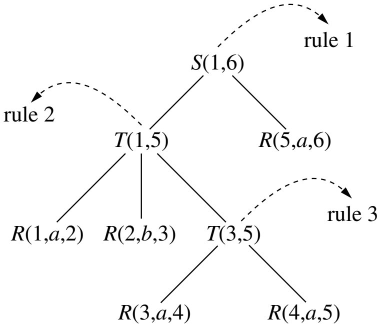
Consider query boolean query $S(1, 6)$ and the following Datalog program

1. $S(x_1, x_3) \leftarrow T(x_1, x_2), R(x_2, a, x_3)$
2. $T(x_1, x_4) \leftarrow R(x_1, a, x_2), R(x_2, b, x_3), T(x_3, x_4)$
3. $T(x_1, x_3) \leftarrow R(x_1, a, x_2), R(x_2, a, x_3)$

and the instance {R(1, a, 2), R(2, b, 3), R(3, a, 4), R(4, a, 5), R(5, a, 6)}.

Top-Down Evaluation

How the proof is generated? (algorithm in Abiteboul & al: Foundations of Databases page 316, Section 13.2 Top-Down Techniques)



(a) Datalog proof

Further Readings

Abiteboul & al: Foundations of Databases

- Section 13.1 Semi-naive Evaluation
- Section 13.3 Magic-Set Evaluation