CS 784: Foundations of Data Management **Spring 2019**

Lecture 8: Introduction to Datalog

Instructor: Paris Koutris

Relational Algebra is the core language for databases, but its expressibility is limited. The most basic problem that is not expressible in RA is the *graph transitive closure*. In the following lectures, we will introduce a new language, called *Datalog*, that allows us to express more complex problems by adding *recursion*. Datalog has seen many applications over the last years, including data integration, declarative networking, and program analysis.

8.1 Datalog Syntax

A *Datalog rule* is an expression of the form

$$R(\vec{x}) : -R_1(\vec{x_1}), \ldots, R_n(\vec{x_n}) \tag{8.1}$$

This is the same syntax as the one we used for conjunctive queries; the big difference is that now we allow for a relation R to appear both on the left and the right side of the rule. A *Datalog program* is just a finite set of Datalog rules.

The schema in Datalog consists of two different types of schemas:

- **Extensional Schema:** this is the set of *extensional relations*: an extensional relation occurs only in the right-hand-side of the rules. Such relations are intuitively the "input" of the Datalog program.
- **Intensional Schema:** this is the set of *intensional relations*: an intensional relation occurs at least once in the left-hand-side of a rule. Intensional relations are the "output" of the Datalog program.

A Datalog program semantically is a mapping from database instances over the extensional schema (which are called EDB) to database instances over the intensional schema (which are called IDB). Let's see some example of Datalog programs below.

Example 8.1. *Let $R(A, B)$ be a relation that contains the edges of a directed graph. The following Datalog program computes the transitive closure of the graph: all the pairs (u, v) of vertices, such that there is a directed path from node u to node v:*

```
T(x,y) :- R(x,y).
T(x,y) :- T(x,z), R(z,y).
```

The second rule is called a linear rule*, because the intensional relation T of the head appears exactly once in the right-hand-side. The following Datalog program, which also computes transitive closure, contains a non-linear rule.*

```
T(x,y) :- R(x,y).
T(x,y) :- T(x,z), T(z,y).
```

Example 8.2. *Let us again assume that $R(A, B)$ describes the edges of a directed graph. We want to write a Datalog program that computes (a) the nodes of the graph such that there exists a cycle of odd length that goes through, (b) the nodes of the graph such that there exists a cycle of even length that goes through, and (c) the nodes of the graph such that there exists a cycle of any length that goes through*

```
OddPath(x,y) :- R(x,y).
EvenPath(x,y) :- R(x,z), OddPath(z,y).
OddPath(x,y) :- R(x,z), EvenPath(z,y).
OddCycle(x) :- OddPath(x,x).
EvenCycle(x) :- EvenPath(x,x).
Cycle(x) :- OddCycle(x).
Cycle(x) :- EvenCycle(x).
```

The relations OddPath and EvenPath are called mutually recursive relations, because they appear on each other's bodies.

8.2 Datalog Semantics

There are 3 different equivalent semantics for Datalog: model-theoretic, fixpoint and proof-theoretic. Here we will discuss only the first two.

8.2.1 Model-Theoretic Semantics

We start by associating a (first-order) logical sentence to each Datalog rule. For example, the rule $\rho : T(x, y) : -T(x, z), R(z, y)$ gives the following logical sentence: $\phi_\rho = \forall x, y, z(T(x, z) \wedge R(z, y) \rightarrow T(x, y))$. In general, for a rule ρ of the form (8.1), we associate the following logical sentence:

$$\phi_r = \forall x_1, \ldots, x_k(R_1() \wedge R_2() \cdots \wedge R_k() \rightarrow R())$$

where $x_1, \ldots, x_k$ are the variables in the body of the rule. An interesting observation is that the logical sentences of the above form are *Horn clauses*: Horn clauses are formulas that consist of a disjunction of literals, where there exists at most one positive literal.

Let Σ_P be the set of logical sentences ϕ_ρ, for every rule ρ in the Datalog program P.

Definition 8.3. *Let P be a Datalog program. A pair of instances (I, J), where I is an EDB, and J is an IDB, is a* model *of P if (I, J) satisfies Σ_P.*

Given an EDB I, the minimal model *of P, denoted $J = P(I)$, is a minimal IDB J such that (I, J) is a model of P.*

We can show that a minimal model always exists, and it is also unique. Also, the minimal model contains only tuples with values from the active domain $\mathbf{adom}(I)$. The semantics of a Datalog program P executed on EDB I is exactly the minimum model $P(I)$.

Exercise 8.4. *Consider the transitive closure on the following instance: $I = \{R(1,2)\, R(2,3), R(3,4)\}$. What is the minimal model in this case? Can you find a non-minimal model and a non-model?*

8.2.2 Fixpoint Semantics

Let P be a Datalog program, and an EDB I. For an EDB J, we say that a fact/tuple t is an *immediate consequence* of I, J if either $t \in I$, or it is the direct result of a rule application on I, J. We define the *immediate consequence operator* for P, denoted T_P, as follows. For every EDB J, $T_P(J)$ contains all the facts that are immediate consequences of I, J.

Lemma 8.5. *The operator T_P is monotone; in other words, if $I \subseteq J$ then $T_P(I) \subseteq T_P(J)$.*

Definition 8.6. *An instance I is a* fixpoint *for T_P if $T_P(I) = I$.*

We can now show the connection of the fixpoint semantics to the model-theoretic semantics.

Theorem 8.7. *For each Datalog program P and EDB I, the immediate consequence operator T_P has a unique, minimal fixpoint J that contains I, which equals the model $P(I)$.*

The fixpoint semantics give us an algorithm that computes the output of a Datalog program. We start with the input I, which is an EDB instance. We then compute $T_P(I)$, then $T_P^2(I)$, and so on. Recall that the operator T_P is monotone. Also, at every iteration we compute at least one new immediate consequence, and there is only a polynomial number of such tuples (since any new tuple must use values from the active domain). Thus, after a polynomial number of steps, we will reach a fixpoint. This way of evaluating Datalog is called the *naive* Datalog evaluation.

Exercise 8.8. *Consider the transitive closure on the following instance: $I = \{R(1,2)\, R(2,3), R(3,4)\}$. Show the application of the operator T_P until it reaches the fixpoint.*

8.3 More on Datalog

Lemma 8.9. *Every Datalog program P is monotone.*

There are many interesting properties one can express in Datalog.

Example 8.10. *Suppose we have a relation $A(x, y)$ that expresses the fact that y is the parent of x. The following Datalog program, called* same generation*, computes the pair (u, v) that have a common ancestor and belong in the same "generation" w.r.t. to the ancestor.*

```
S(u,v) :- A(u,x), A(v,x).
S(u,v) :- A(u,x), S(x,y), A(v,y).
```

A more modern application of Datalog is in program analysis, and in particular points-to analysis [PT]. In this setting, we are given a program (in any programming language), and we want to compute what points to what.

Example 8.11. *This example describes the Datalog rules for a simple type of points-to analysis in C programs, called Andersen's analysis. Initially, we turn instructions in C to predicates in Datalog:*

- `y=&x` *: AddressOf(y, x)*
- `y=x` *: Assign(y, x)*
- `y=*x` *: Load(y, x)*
- `*y=x` *: Store(y, x)*

We want to compute the relation PointsTo(y,x), i.e. whether variable y may point to the location of variable x. We can do this using the following Datalog program:

```
PointsTo(y,x) :- AddressOf(y,x).
PointsTo(y,x) :- Assign(y,z), PointsTo(z,x).
PointsTo(y,w) :- Load(y,x), PointsTo(x,z), PointsTo(z,w).
PointsTo(z,w) :- Store(y,x), PointsTo(y,z), PointsTo(x,w).
```

References

[Alice] S. ABITEBOUL, R. HULL and V. VIANU, "Foundations of Databases."

[PT] Y. SMARAGDAKIS, and G. BALATSOURAS, "Pointer Analysis.", *Foundations and Trends in Programming Languages, 2015.*