CS 838: Foundations of Data Management **Spring 2016**

Lecture 9: Datalog with Negation

Instructor: Paris Koutris 2016

In this lecture we will study the addition of negation to Datalog. We start with an example.

Example 9.1. *Suppose that we want to compute the complement of graph transitive closure (i.e. output all edges (a, b) such that there is no directed path from a to b). We can compute this using the following Datalog program:*

```
V(x)    :- R(x,y).
V(y)    :- R(x,y).
T(x,y)  :- R(x,y).
T(x,y)  :- T(x,z), R(z,y).
TC(x,y):- V(x), V(y), not R(x,y).
```

Notice that we now allow negation as part of the rules. But what are the semantics when we add negation to Datalog? We investigate this next.

The first attempt to define the semantics would be by extending the fixpoint semantics. The immediate consequence operator can still be easily defined (it will be a relational algebra query). But it turns out that a minimal fixpoint is not well-defined in the case of Datalog with negation! Consider the following examples:

- $R(x) :- S(x), \neg R(x)$ has no fixpoint!
- $R(x) :- S(x)\neg T(x)$ and $T(x) :- S(x)\neg R(x)$. Then there are two minimal fixpoints!
- There are also Datalog programs where the iterative computation does not converge, or it does converge to a fixpoint that is not the minimal one!

The model-theoretic approach also fails to work for Datalog. So how can we define the semantics of Datalog with negation? We will see several different ways that we can do this.

9.1 Semi-positive Datalog

In *semi-positive Datalog*, we are allowed to have negation only in front of edb relations. To define the semantics for this fragment, we just interpret $\neg R(\vec{x})$ as follows: $\neg R(\vec{x})$ is true if and only if $\vec{x}$ is in the active domain and $\vec{x}$ is not in R. The standard iterative computation works for semi-positive Datalog, since the negated edb relations can essentially be replaced by a positive relation (which is the complement). Here is an example semi-positive Datalog program:

```
RC(x,y) :- V(x), V(y), not R(x,y).
T(x,y)  :- RC(x,y).
T(x,y)  :- T(x,z), RC(z,y).
```

9.2 Stratified Datalog

A stratified Datalog program with negation is a generalization of the idea behind semi-positive Datalog. In semi-positive Datalog, the negation is only in front of edb relations. We can now see that we can also add negation in front of an idb that is the output of a semi-positive program, and so on. Let's define this more formally.

Definition 9.2. *A* stratification *of a Datalog program P is a partition of P into Datalog sub-programs $P^1, \ldots, P^m$ such that:*

- *All the rules defining an idb relation are in the same partition.*
- *If R appears positive in the body of a rule with head R', R should be defined in a partition before or where R' is defined.*
- *If R appears negated in the body of a rule with head R', R should be defined in partition strictly before where R' is defined.*

Not every Datalog program can be stratified. If a Datalog program can the stratified, this leads to well-defined semantics: we start from the first stratum (partition), which will be a semi-positive program, compute the result, and then view the idb relations as edb relations in the second stratum, and so on.

How do we determine if a Datalog program is stratifiable? We construct the *precedence graph*: whenever idb R appears positive in a rule with head S, we add a directed edge (R, S) with label $+$. Whenever R appears negative in a rule with head S, we add an edge (R, S) with label $-$.

Lemma 9.3. *A Datalog program with negation can be stratified if and only if the precedence graph has no directed cycles with a negative edge.*

We know that a Datalog program can have many possible stratifications. It turns out that all possible stratifications lead to the same result, independent of the stratification order.

9.3 Well-Founded Semantics

The well-founded semantics classify each possible answer to facts that are true, false, and facts that are *unknown*. In other words, we want to find a *3-valued* model to describe the answer to the program. Formally, each possible fact in the active domain will be mapped to one of three values:

$0, 1, 1/2$ (0 means certainly not in the output, 1 means certainly in the output, and $1/2$ means unknown/uncertain). This allows us to define semantics for all Datalog programs with negation.

We define next the *win-move* game. In this game, we have a directed graph desctibed by the relation $moves(x, y)$. Two players make alternatively moves in the graph according to the relation $moves$. A player loses if she has no moves to make. We want to compute the following predicate $win(x)$, which means that the player has a winning strategy if she moves from vertex x. The following Datalog program defines the relation:

$$win(x) : -moves(x, y), \neg win(y),$$

Example 9.4. *Consider the following instance of the relation moves:*

$$(1, 2), (2, 3), (3, 1), (1, 4), (4, 5), (4, 6), (6, 7).$$

The fixpoint for the idb win is the following 3-valued instance: $7 : 0, 6 : 1, 5 : 0, 4 : 1, 1 : 1/2, 3 : 1/2, 2 : 1/2$.

If we define now the immediate consequence operator by using the 3-valued semantics, we can show that there exists a least fixpoint for every Datalog program with negation.

References

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