

CS589 Principles of DB Systems

Fall 2008

Lecture 4d: Recursive Datalog with Negation –
What is the query answer defined to be?

Lois Delcambre

imd@cs.pdx.edu

503 725-2405



Goals for today

- Introduce the problem with recursion and negation in Datalog.
- Introduce stratification – a syntactic restriction that avoids the problem.
- Mention several ways to define the semantics of a Datalog program.

Several Datalog languages

This one is often referred to as "Datalog"; it was defined first.

This one is often referred to as "Datalog \neg " or "Datalog-neg"

- Datalog – multiple rules, no negation, with recursion. *Recursion but not relationally complete.*
- Datalog – multiple rules, with negation, no recursion. *Relationally complete but no recursion.*
- Datalog – multiple rules, with negation, with recursion. *Relationally complete with recursion but some queries are ambiguous!*

Our focus for QL equiv.

Our focus now

Datalog program and its dependency graph

Given a DB with two relations:

Topics(Topic) and Interests(Person,Topic)

What is this query computing?

Result(a):-Interests(a,b), \neg DIFF(a).

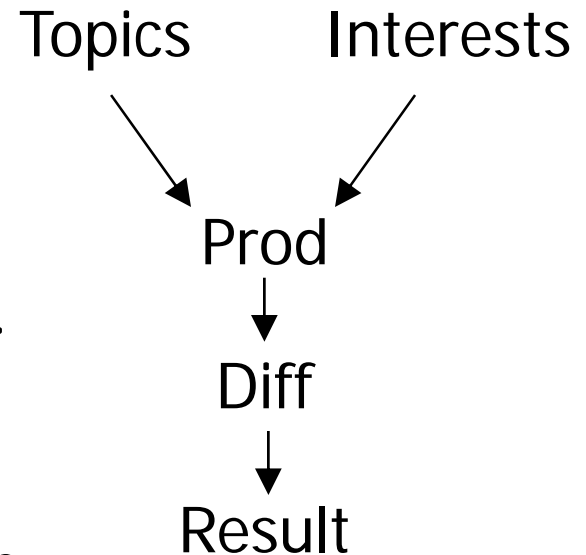
Diff(a) :- Prod(a,b), \neg Interests(a,b).

Prod(a,b) :- Interests(a,c), Topics(b).

Create one node for each body predicate.

Draw an arrow from body predicate
to corresponding head predicate.

Acyclic dependency graph \equiv not recursive.



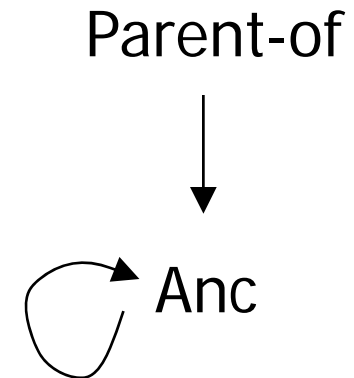


Another example

$\text{Anc}(x,y) \text{ :- Parent-of}(x,y).$

$\text{Anc}(x,z) \text{ :- Anc}(x,y), \text{Parent-of}(y,z).$

Recursive Datalog program
has a cycle in the dependency
graph.





Now consider Datalog with recursion & negation

Person(Dan). (one tuple in base relation)

Student(x) :- Person(x), \neg Employee(x).

Employee(x) :- Person(x), \neg Student(x).

What is the query answer?

Either Student(Dan) or Employee(Dan),
depending on which rule fires first.

Two different query answer!



We have a problem – for any data.

Student(x) :- Person(x), \neg Employee(x).

Employee(x) :- Person(x), \neg Student(x).

Imagine we have some number of persons in the database.

They will ALL end up at Students or as Employees, depending on which rule fires first.

Two different query answers!

What if the Person table is empty?

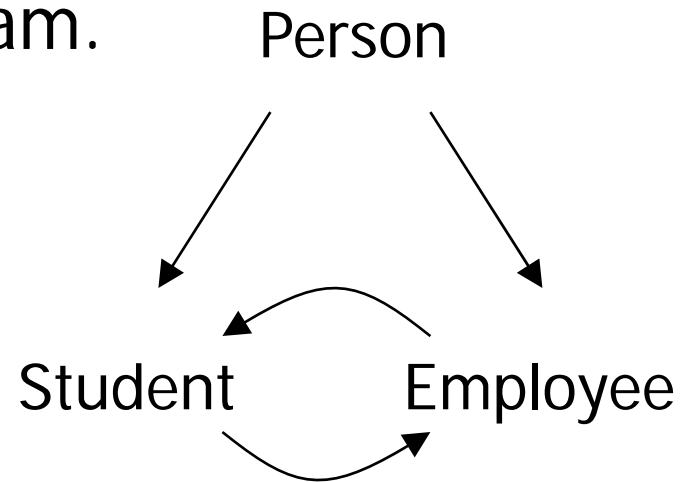
What does the dependency graph look like?

Person(Dan). (one tuple in base relation)

Student(x) :- Person(x), \neg Employee(x).

Employee(x) :- Person(x), \neg Student(x).

This is a recursive program.



More about this dependency graph

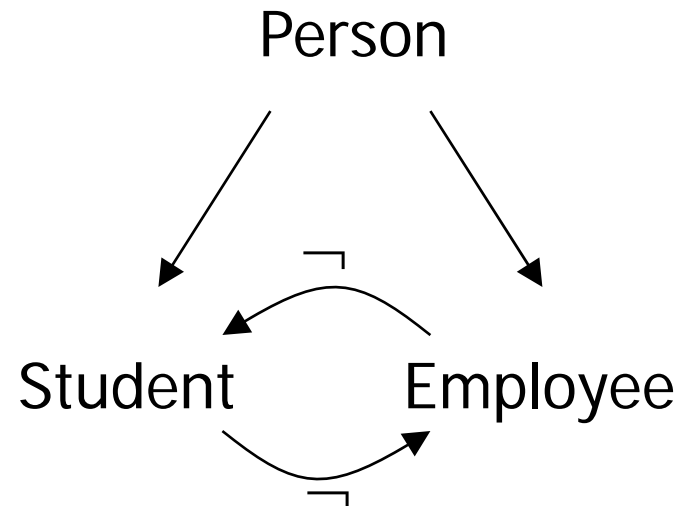
Person(Dan). (one tuple in base relation)

Student(x) :- Person(x), \neg Employee(x).

Employee(x) :- Person(x), \neg Student(x).

Cycle in dependency graph \rightarrow recursion.

Label negative predicates in dependency graph.



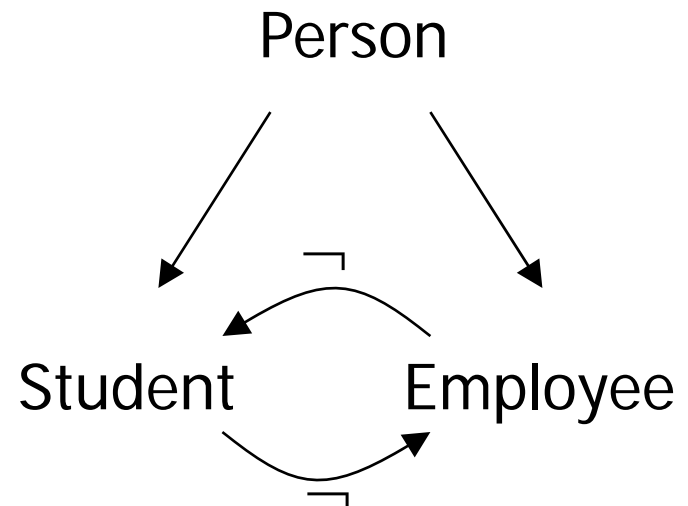
Now consider recursion & negation

Person(Dan). (one tuple in base relation)

Student(x) :- Person(x), \neg Employee(x).

Employee(x) :- Person(x), \neg Student(x).

We have a problem when
a cycle in a dependency
graph includes (at least)
one negative edge.



Cycle with one (or more) negative edges

Person(Dan). (one tuple in base relation)

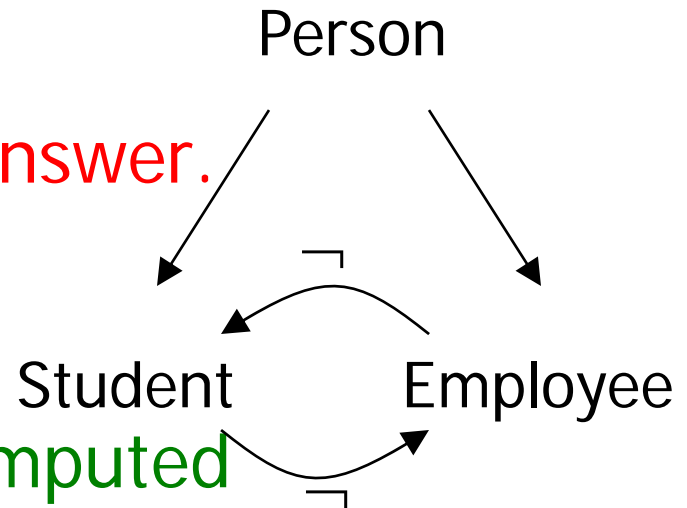
Student(x) :- Person(x), \neg Employee(x).

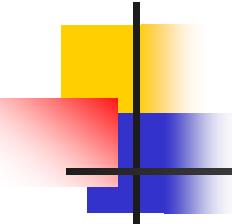
Employee(x) :- Person(x), \neg Student(x).

Recursion (for Student, say) is attempting to add tuples to answer.

But it is subtracting Employee tuples (to find the Students).

And ... Employees are being computed recursively at the same time.





Another example with negation & recursion

Node(a).

Node(b).

Node(c).

Node(d). **These are facts from the database.**

Arc(a,b).

Arc(c,d).

Reachable(a).

Reachable(y) :- Reachable(x), Arc(x,y).

Unreachable(x) :- Node(x), \neg Reachable(x).



Do we have a problem?

...

Arc(c,d).

Reachable(a).

Reachable(y) :- Reachable(x), Arc(x,y).

Unreachable(x) :- Node(x), \neg Reachable(x).

What if the second rule fires first? And then the first rule fires?

What if the first rule fires repeatedly ... and THEN we fire the second rule?

Stratification – for Datalog with negation and recursion

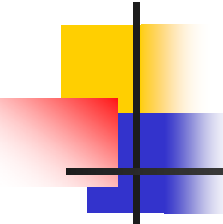
A stratification of a Datalog program P is a partition of P into strata or layers where, for every rule, $H :- A_1, \dots, A_m, \neg B_1, \dots, B_q$

- The rules that define the predicate H are all in the same strata.
- For all positive predicates in this rule, their strata is less than or equal to the strata of this rule.
 $\text{Strata}(A_i) \leq \text{Strata}(H)$.
- For all negative predicates in this rule, their strata is strictly less than the strata of this rule.
 $\text{Strata}(B_j) < \text{Strata}(H)$.



Intuition behind stratification

- In any strata, compute a predicate – in its entirety.
- If you EVER use a predicate negatively, then make sure it is **ALREADY, COMPLETELY** computed.
- So ... stratification prevents you from using recursion at the same time you're computing a predicate that is used negatively.



Exercise: Define a stratification for this datalog program.

Node(a).

Node(b).

Node(c).

Node(d).

Arc(a,b).

Arc(c,d).

Reachable(a).

Compute all Reachable facts
(positively) in Strata 1.

Then, use Reachable
negatively in Strata 2.

Strata 1 Reachable(y) :- Reachable(x), Arc(x,y).

Strata 2 Unreachable(x) :- Node(x), \neg Reachable(x).



Can you define a stratification for this program?

Person(Dan). (one tuple in base relation)

Student(x) :- Person(x), \neg Employee(x).

Employee(x) :- Person(x), \neg Student(x).



Comments

- Stratified Datalog programs always have a unique answer.
- Note that Stratification forces the choice of one of various possible answers.
- Not all Datalog programs can be stratified.
- Some programs can be stratified in more than one way. If so, then all stratifications will produce the result.



Semantics of a Datalog Program

- The fixed point of the “immediate consequence” operator, applied to the Datalog program.
- The minimal model for the Datalog program. (We’ll talk about models and how they relate to relational databases on Wednesday.)

Plus others, particularly for Datalog with negation.



Comments

- For Datalog with recursion, but NO negation, then:
 - The minimal model is unique.
 - The minimal model is always the intersection of all the models.
 - The minimal model is the same as the fixed point of the immediate consequence operator.
 - This language is monotonic (rules only add facts)
- For Datalog with recursion & negation:
 - There may not be a unique minimal model.



Questions

- If a predicate appears in the head of one rule and in the body of another, is the Datalog program necessarily recursive?
- If a datalog program is not recursive, how many times do we have to fire the rules?