Introduction to Logic and Databases

R. Ramakrishnan

Database Management Systems, Gehrke-Ramakrishnan, 3rd ed

Prolog Example

$$\forall x, y, z \begin{pmatrix} p(x, y) :- e(x, y) \\ p(x, y) :- e(x, z), p(z, y) \end{pmatrix}$$

$$p(\sigma, ?)$$

Syntax:

Rule	h:- b.
Literal	$p(x,y); q([5 \mid x], z)$
Tuple	$p(5,6)$
Term	$[5 \mid x]; 5; x; f(5,y)$
Predicate	$p; e; "+"; "\times" \ldots$

Why, and what

- Idea
  - If you want to compute a set of facts (tuples), just describe the set, and let the DBMS/Compiler worry about efficiency.
- Describing *what* you want
  - LOGIC PROGRAM
    set of RULES + FACTS
- Semantics
  - Intuitively
    Head of rule is true if (and only if) body is true.
  - Formally
    The Least Model of program

Model

- An '<u>Interpretation</u>'
  - A mapping: Predicate name → Set of Facts
- A '<u>Model</u>'

  An interpretation **I** such that:
  - ∀ "evaluable" predicate p, $\mathbf{I}(p)=natural(p)$
  - ∀ Rule $p(t)\text{:-}q_1(t_1),\ldots,q_n(t_n)$:
    For any assignment σ of constants to variables,
    $\sigma(t)$ is in $\mathbf{I}(p)$ IF $\forall i$, $\sigma(t_i)$ is in $\mathbf{I}(q_i)$
- <u>Least Model</u>
  - A model **I** such that for every other model **I'**,
    ∀ predicate p, $\mathbf{I}(p) \subseteq \mathbf{I}'(p)$

Example

1. par(abel, adam).
2. par(abel, eve).
3. par(sem, abel).
4. anc(x,y):- anc(x,z),anc(z,y).
5. anc(x,y):- par(x,y).
6. gen(x,I):- gen(y,J), par(x,y), I=J+1.
7. gen(adam,1)

Evaluate predicate "+": +(I, J, K) ≡ I=J+K

Least Model (Rules 1-5)

par(abel, adam)	anc(abel, adam)
par(abel, eve)	anc(abel, eve)
par(sem, abel)	anc(sem, abel)
	anc(sem, adam)
	anc(sem, eve)

If add anc(sem, sem), is this a Model?

Applying a rule

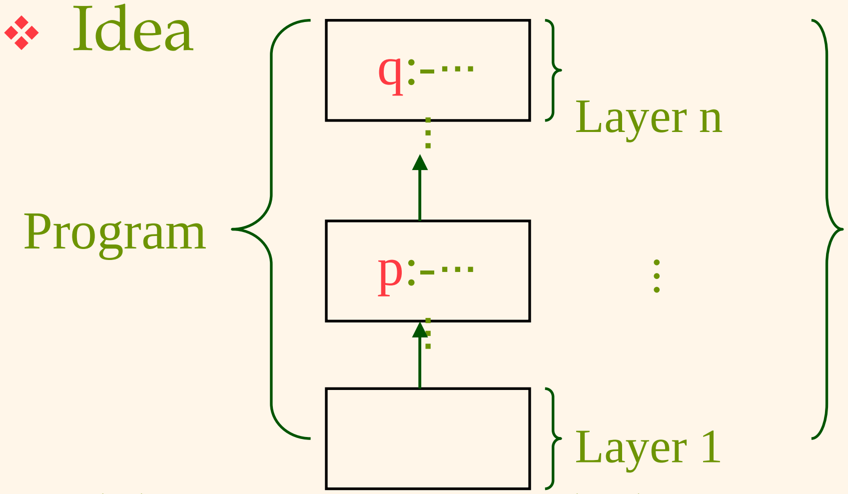
- If rule R and a set of facts F are given, the rule can be used to generate new facts.
- Operator T_p

  $Tp(F) = F \cup$ {all facts that can be generated in 1 step by applying rules of program P}
- Fixpoints of T_p

  A set of facts $\boldsymbol{F}$ is a fixpoint of T_p if $\boldsymbol{F = T_p(F)}$
- Key Results
  1. If P is a Horn Clause program (w/o negation), there is a unique least fixpoint (obtained by starting with the empty set of facts, and applying T_p)
  2. Least Fixpoint = Least Model

Stratification

- Does every program have least model?
  - Not if there is NEGATION!
  - $p \text{:-} \neg q.$ $\quad$ $q \text{:-} \neg p.$
  - 2 least models: (p False, q True) and (p True, q False)
- Stratification
  - $p \rightarrow q$: $\exists$ some rule with head q, and p in body.
  - $p \rightarrow^* q$: $\exists$ some predicate s: $p \rightarrow s$, $s \rightarrow^* q$, or $p \rightarrow q$
  - If $\neg p \rightarrow^* q$, then $q \nrightarrow^* p$ and $\neg q \nrightarrow^* p$
- Idea



Compute from layer 1 to n:
For layer i, Predicates from layers (1…i-1) essentially BASE predicates.

Stratification – Cont.

- Dependency Graph
  - A NODE for each predicate.
  - An ARC (directed) from p to q if there is rule with head predicate q and body predicate p.
  - Note:
    - For simplicity, we don't consider programs with '$\vee$' in rule heads.
    - If p is NEGATED, the arc is labeled '$\neg$'
- A Program is STRATIFIED if the dependency graph has no cycle passing through an arc labeled '$\neg$'.

Stratification - Cont.

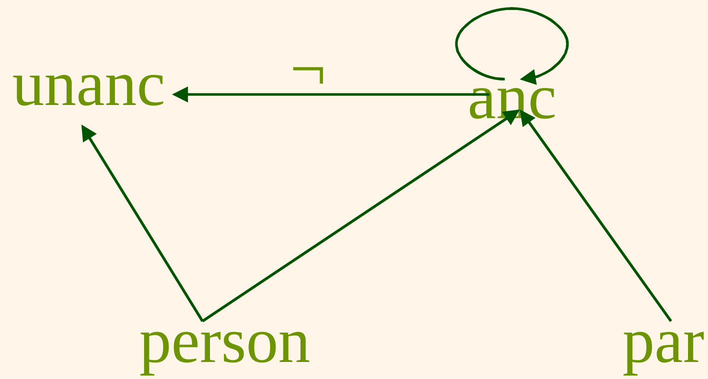
- Strata
  - Merge all nodes in cycles with no ¬ arcs to obtain a DAG. Each node in the DAG is called a STRATUM or LAYER.
  - Without loss of generality, assume a topological sort that orders all strata: L_1, L_2, …, L_n. Thus, no predicate in L_i is used to define a predicate in L_j if i>j.
- Idea
  - Evaluate layer-by-layer, from L_1 to L_n. (Least Fixpoint evaluation within each layer.)
  - Guaranteed to lead to a unique result.
- Question
  - Which has higher priority for minimization, L_1 or L_n?

Stratified Models

- If predicate p is in Li, q is in Lj, I > j, then every p-fact has lower priority than any q fact.
- Model N is "smaller" than model M if for each fact A in N-M, there is a fact B in M-N such that A has lower priority than B.
- Theorem: If P is a stratified program
  - There is a "least" model M for P
  - M = layer-by-layer least fixpoint

Example

unanc(x,y):- person(x), person(y), ¬anc(x,y).
anc(x,y):- anc(x,z), par(z,y).
anc(x,x):-person(x).



p(s):- ¬ q(s).
q(s):- ¬ p(s).

p(0).
p(x):- ¬p(s(x)).

even(0).
even(s(x)):- ¬even(x).

Negation

- Prolog:
  - Negation as failure.
  - Why isn't this appropriate?
- Example

rich(x) ∨ tv_evangelist(x) :- famous(x).
famous(Forbes).
Prof(Raghu).

There are 2 minimal models. Note that '∨' is treated as an exclusive or; this is a consequence of minimal model semantics.

rich(x) :- famous(x), ¬ tv_evangelist(x)
famous(Forbes).
prof(Raghu).

Which minimal model is better? Intuitively, we place a higer priority on minimizing the relation tv_enangelist. (The fewer the better).

Properties of a query

- Safety
  - Finite set of answers.

    e.g.   >(x,5)?   (unsafe)

    e.g.   like(x,y):- nice(x).

    nice(john).

    like(john,u)?   (safe)

    plus

    good_sal(z):- z>40k, z<60k.

    like(john,u)?   (unsafe)

- Range-Restriction
  - For every rule, every variable x in head also appears in positive body literal.

    This a sufficient condition for safety of Datalog programs.
  - Definition: DATALOG - No function symbols or arithmetic.