Introduction to Logic and Databases

R. Ramakrishnan

Database Management Systems, Gehrke-Ramakrishnan, 3rd ed
Prolog Example

∀x, y, z
\[
\begin{align*}
p(x, y) & : \neg e(x, y) \\
p(x, y) & : \neg e(x, z), p(z, y)
\end{align*}
\]

\[p(\sigma, ?)\]

Syntax:

Rule  \[h:- b.\]

Literal  \[p(x,y); q([5|x],z)\]

Tuple  \[p(5,6)\]

Term  \[[5|x]; 5; x; f(5,y)\]

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 ‘Interpretation’**
  - A mapping: Predicate name → Set of Facts

- **A ‘Model’**
  An interpretation I such that:
  - ∀ “evaluable” predicate p, \( I(p) = \text{natural}(p) \)
  - ∀ Rule \( p(t):-q_1(t_1),...,q_n(t_n) \):
    For any assignment \( \sigma \) of constants to variables, \( \sigma(t) \) is in \( I(p) \) IF ∀i, \( \sigma(t_i) \) is in \( I(q_i) \)

- **Least Model**
  - A model I such that for every other model I’, ∀ predicate p, \( I(p) \subseteq I’(p) \)
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** \( T_p \)
  \[ T_p(F) = F \cup \{ \text{all facts that can be generated in 1 step by applying rules of program } P \} \]
- **Fixpoints of** \( T_p \)
  A set of facts \( F \) is a fixpoint of \( T_p \) if \( F = T_p(F) \)
- **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**!

\[ p \leftarrow \neg q, \quad q \leftarrow \neg p. \]

2 least models: (p False, q True) and (p True, q False)

Stratification

\[ p \rightarrow q : \exists \text{ some rule with head } q, \text{ and } p \text{ in body.} \]
\[ p \rightarrow *q : \exists \text{ some predicate } s: p \rightarrow s, s \rightarrow *q, \text{ or } p \rightarrow q \]

If \( \neg p \rightarrow *q \), then \( q \rightarrow *p \) and \( \neg q \rightarrow *p \)

Idea

Program

\[ q \leftarrow \cdots \]
\[ p \leftarrow \cdots \]

Layer n

Layer 1

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 ‘√’ in rule heads.
    - If p is NEGATED, the arc is labeled ‘¬’

- A Program is **STRATIFIED** if the dependency graph has no cycle passing through an arc labeled ‘¬’.
Stratification - Cont.

- **Strata**
  - Merge all nodes in cycles with no \( \neg \) 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: \( L_1, L_2, \ldots, L_n \). 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 $L_i$, $q$ is in $L_j$, $i > j$, 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

\[
\begin{align*}
\text{rich}(x) \lor \text{tv\_evangelist}(x) & : - \text{famous}(x). \\
\text{famous}(\text{Forbes}). \\
\text{Prof}(\text{Raghu}).
\end{align*}
\]

There are 2 minimal models. Note that ‘\lor’ is treated as an exclusive or; this is a consequence of minimal model semantics.

\[
\begin{align*}
\text{rich}(x) & : - \text{famous}(x), \neg \text{tv\_evangelist}(x) \\
\text{famous}(\text{Forbes}). \\
\text{prof}(\text{Raghu}).
\end{align*}
\]

Which minimal model is better? Intuitively, we place a higher priority on minimizing the relation tv\_evangelist. (The fewer the better).
Properties of a query

- Safety
  - Finite set of answers.

  e.g. \( >(x,5)? \) (unsafe)
  e.g. \( \text{like}(x,y):- \text{nice}(x). \)  
    \( \text{nice}(\text{john}). \)
    \( \text{like}(\text{john},u)? \) (safe)

  plus
  \( \text{good}_{-}\text{sal}(z):- z>40k, z<60k. \)
  \( \text{like}(\text{john},u)? \) (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.