Introduction to Logic and Databases

R. Ramakrishnan

Database Management Systems, Gehrke-Ramakrishnan, 3rd ed

Prolog Example

\[
\begin{align*}
\forall x, y, z & \left( 
\begin{array}{c}
p(x, y) \leftarrow \neg \neg e(x, y) \\
\neg \neg e(x, y, z), p(z, y) \\
p(\sigma_7) 
\end{array} \right) \\
p(x, y) & \leftarrow e(x, y) \\
p(x) & \leftarrow \neg \neg e(x, y)
\end{align*}
\]

Syntax:

Rule \( h \leftarrow b \).
Literal \( p(x, y) \neg \neg \{S \} \).
Tuple \( p(5, 6) \).
Term \( \{S \} \). 5: \( x \neg \neg \{5, y \} \).

Predicate \( p : e : \times \neg \neg + \); “\( \times \)…"

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:
    - \( \forall \) “evaluable” predicate \( p \).
    - \( \forall \) Rule \( p(t_0) \leftarrow q(t_1), \ldots, q(t_n) \).
  - For any assignment of constants to variables, \( \sigma(t) \) is in \( I(p) \) if \( \forall \theta \) \( \sigma(t) \) is in \( I(q) \).

- **Least Model**
  - A model I such that for every other model \( I' \), \( \forall \) predicate \( p \), \( I(p) \subseteq I'(p) \)

Example

1. \( \text{par}(abel, adam) \).
2. \( \text{par}(abel, eve) \).
3. \( \text{par}(sem, abel) \).
4. \( \text{anc}(x,y) \leftarrow \text{anc}(x,z), \text{anc}(z,y) \).
   - Evaluate predicate “\( \times \)” \( + \{I, J, K \} \equiv I+J+K \).

   Least Model (Rules 1-5): \( \text{par}(abel, adam) \), \( \text{par}(abel, eve) \), \( \text{anc}(abel, abel) \), \( \text{anc}(sem, abel) \), \( \text{anc}(sem, adam) \), \( \text{anc}(sem, eve) \).

   If add \( \text{anc}(sem, sem) \), 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** \( T_F \)
  \( T_F(F) = F \cup \{ \text{all facts that can be generated in 1 step by applying rules of program P} \} \)

- **Fixpoints of** \( T_F \)
  - A set of facts \( F \) is a fixpoint of \( T_F \) if \( F = T_F(F) \)

- **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 \( T_F \)).
  2. Least Fixpoint = Least Model
Stratification

Does every program have least model?
Not if there is NEGATION!
\( p \rightarrow \neg q \), \( q \rightarrow \neg p \).
2 least models: \( p \) False, \( q \) True and \( p \) True, \( q \) False

Stratification
\( p \rightarrow q \) : some rule with head \( q \) and \( p \) in body.
\( p \rightarrow \neg q \) : some predicate \( s \) : \( p \rightarrow s \rightarrow q \), or \( p \rightarrow \neg q \)
If \( p \rightarrow q \) then \( q \rightarrow p \) and \( \neg q \rightarrow \neg p \)

Idea
Compute from layer 1 to \( n \):
For layer \( i \), predicates from layers \( (1 \ldots i-1) \) essentially BASE predicates.

Program

Layer 1
Layer n

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 ‘\(-\)’.

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

Negation

Prolog:
Negation as failure.
Why isn’t this appropriate?

Example
\[ \text{rich}(x) \lor \text{tv_evangelist}(x) \rightarrow \text{famous}(x). \]
\[ \text{famous}(\text{Forbes}). \]
\[ \text{prof}(\text{Raghu}). \]

There are 2 minimal models. Note that \( \lor \) is treated as an exclusive or; this is a consequence of minimal model semantics.
\[ \text{rich}(x) \rightarrow \text{famous}(x), \text{tv_evangelist}(x) \]
\[ \text{famous}(\text{Forbes}). \]
\[ \text{prof}(\text{Raghu}). \]

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. 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.