COMP353 Databases

Logical Query Languages:

Datalog

Logical Query Languages

- Motivation
  - Logical ***if-then*** rules extend rather “naturally” and easily to recursive queries; Relational algebra doesn’t!
    - Recursion is considered in SQL3
  - Logical rules (Datalog) form a basis for development of many concepts and techniques in database and knowledge base systems, with many applications such as *data integration*

Datalog

AlongMovie(Title, Year) ← movie(Title, Year, Length, Type), Length >= 100.

- The **head** – the left hand side of the arrow/implication
- The **body** – the right hand side is a conjunction (AND) of predicates (called subgoals)
- NOTE: The book uses **AND** in the rule bodies instead of commas.
- The head is a positive predicate (atom) and the subgoals in the rule body are *literals* (an atom or a negated atom)
  - Atom - a formula of the form $p(T_1,\ldots,T_n)$, where **p** is a predicate name and $\mathbf{T_i}$'s are *terms*
  - Predicate – **normal** (ordinary) relation name (e.g., movie, p) or

    built_in predicates (e.g., $\geq$ in the above example)
  - Terms (arguments) – In Datalog, T_i is a *variable* or a *constant*
  - Subgoals in the rule body may be negated using NOT

Datalog

longMovie(Title, Year) ← movie(Title, Year, Length, Type), Length >= 100.

- A variable in a rule body is called *local* if it appears only in the rule body, e.g., Length and Type
- The head is true if there are values for local variables that make every subgoal (in the rule body) true
- If the body includes no negation, then the rule can be viewed as a join of relations in the rule body followed by a projection on the head variable(s)

Datalog

longMovie(Title, Year) ← movie(Title, Year, Length, Type), Length >= 100.

- This rule may be expressed in RA as:

$$\rho_{\text{longMovie}}(\pi_{\text{Title,Year}}(\sigma_{Length \geq 100}(\text{movie})))$$

Variable-Based Interpretations of Rules

- In principle, given the rule
- r: H $\leftarrow$ B1,…,Bk**.**

  we consider all possible assignments $\mathcal{I}$ of values (constants in the domain) to the variables in the rule.
- Such assignments $\mathcal{I}$ are called interpretations.
- For every interpretation $\mathcal{I}$ of *the rule*, if the body is true under $\mathcal{I}$, we add to the head relation, the tuple defined by H under $\mathcal{I}$.

  (we only consider *ground* interpretations/substitutions).
  - That is, if $\mathcal{I}$(Bi) is true, $\forall$ **i** $\in$ {1,…,k}, then $\mathcal{I}$(H) is true.
  - In this case, we say that "*$\mathcal{I}$ satisfies r" or "$\mathcal{I}$ is a model for r"*, *(this is denoted as $\mathcal{I} \models r$)*

Example

$s(X, Y) \leftarrow r(X, Z), r(Z, Y),$ **NOT** $r(X, Y)$**.**

Instance r:

A	B
1	2
2	3

Instance s:

A	B
1	3

- The only assignments that make the first subgoal true are:
  1. $\mathcal{I}_1$: $X \rightarrow 1, Z \rightarrow 2$
  2. $\mathcal{I}_2$: $X \rightarrow 2, Z \rightarrow 3$.
- In case (1),
  - $Y \rightarrow 3$ makes the second subgoal r(Z,Y) true
  - Since (1, 3) $\notin$ r, then "**NOT** r(X,Y)" is also true
  - Thus, we infer tuple (1, 3) for the head relation, s
- In case (2),
  - No value of "Y" makes the second subgoal true

Tuple-Based Interpretations of Rules

- Consider tuple variables for each positive normal subgoals that range over their relations
  - For each assignment of tuples to each of these subgoals, we determine the implied assignment θ of values to variables
  - If the assignment $\mathcal{I}$ is:
    - consistent and also
    - satisfies all the subgoals (normal and built-ins) in the body

    then we add to the head relation, the tuple defined by the head H under $\mathcal{I}$

Example

$s(X, Y) \leftarrow r(X, Z), r(Z, Y),$ **NOT** $r(X, Y)$.

Instance r:

A	B
1	2
2	3

Instance s:

A	B
1	3

- Have 4 assignments of tuples to subgoals:

  r(X,Z) r(Z,Y)

  1. (1, 2) (1, 2)
  2. (1, 2) (2, 3)
  3. (2, 3) (1, 2)
  4. (2, 3) (2, 3)
- Only the second assignment
  - is consistent for the value assigned to Z and satisfies the negative subgoal "**NOT** r(X,Y)"

  ➔ (1,3) is the only tuple we get for s

Datalog Programs

- A datalog program is a finite collection of rules
- Note: while standard datalog does not allow negation, in our presentation here, the programs and rules are actually in datalog extended with negation and built-in predicates.
- Predicates/relations can be divided into two classes
  - **EDB** Predicates (input relations), also called ***FACTS***
    - Extensional database = relations stored explicitly in DB
  - **IDB** Predicates (derived/output relations), defined by rule(s)
    - Intensional database
    - They are similar to views in relational databases
- **Note: EDB** predicates appear only in the rule body and

  IDBs appear in the head and possibly in the body

Operations in Datalog

- The usual set operations
- **Consider relation schemas r(X,Y) and s(X,Y)**
  - **Intersection**
    - RA: $Q = r \cap s$
    - Datalog: $q(X,Y) \leftarrow r(X,Y),\ s(X,Y).$
  - **Union**
    - RA: $Q = r \cup s$
    - Datalog: the following two rules:
      1. $q(X,Y) \leftarrow r(X, Y)$ **.**
      2. $q(X,Y) \leftarrow s(X, Y)$ **.**
  - **Difference**
    - RA: $Q = r - s$
    - Datalog: $q(X,Y) \leftarrow r(X,Y)$, **NOT** $s(X,Y)$**.**

Operations in Datalog

- Projection operation
  - RA: $p = \pi_x (r)$
  - Datalog: $p(X) \leftarrow r(X,Y).$

Operations in Datalog

- Selection operation
  - RA: $s = \sigma_{X > 10 \textbf{ AND } y = 5}(r)$
  - Datalog: $s(X,Y) \leftarrow r(X,Y), X > 10, Y = 5.$

Operations in Datalog

- Selection operation. Recall the schema of r(X,Y).
  - RA: $s = \boldsymbol{\sigma}_{x > 10 \textbf{ OR } y = 5}(r)$
  - Datalog: the following two rules:
    1. s(X, Y) ← r(X, Y), X >10.
    2. s(X, Y) ← r(X, Y), Y = 5.

    Note: the following Datalog program is equivalent to the above.
    1. s(A,B) ← r(A,B), A>10.
    2. s(C,D) ← r(C,D), D = 5.

Operations in Datalog

- Cartesian Product operation
- Consider relation schemas   r(A, B) and s(C, D)

  - RA: $Q = r \times s$
  - Datalog: $q(X, Y, Z, W) \leftarrow r(X,Y), s(Z,W).$

Operations in Datalog

- Join operation
  - Theta-join with an AND condition, e.g., "c_1 AND c_2"
    - RA: $tj1 = r \bowtie_{X > Z \text{ AND } Y < W} s$
    - Datalog: $tj1(X, Y, Z, W) \leftarrow r(X,Y), s(Z,W), X > Z, Y < W.$
  - Theta-join with an OR condition, e.g., "c_1 OR c_2"
    - RA: $tj2 = r \bowtie_{X > Z \text{ OR } Y < W} s$
    - Datalog: the following two rules:

$$tj2(X, Y, Z, W) \leftarrow r(X,Y), s(Z, W), X > Z.$$

$$tj2(X, Y, Z, W) \leftarrow r(X,Y), s(Z,W), Y < W.$$

Operations in Datalog

- Join operation
  - Equi-join
    - RA: $ej3 = r \bowtie_{Y=Z} s$
    - Datalog: $ej3(X,Y,Z,W) \leftarrow r(X,Y), s(Z,W), Y = Z.$

    OR even better (simpler):

    $ej3(X,Y,Y,W) \leftarrow r(X,Y), s(Y,W).$

Operations in Datalog

- Join operation
  - Natural join
    - RA: nj4 = r $\triangleright\!\triangleleft$ s
    - Datalog: nj4(X,Y,W) $\leftarrow$ r(X,Y), s(Y,W).

Example:Datalog Queries/Programs

- Database schema:
  movie(Title, Year, Length, FilmType, StudioName)
  starsIn(Title, Year, StarName)
- Query: Find the **names** of stars of movies that are at least 100 minutes long
- Relational Algebra Expression:

  $$\mathbf{Q = \pi_{\mathit{starName}}(\sigma_{\mathit{length \geq 100}}(movie) \bowtie starsIn)}$$

- Datalog program:
  r1(Title, Year, Length, Type, Studio) ←
  movie(Title, Year, Length, Type, Studio), Length >= 100.
  r2(Title, Year, Length, Type, Studio, Name) ←
  r1(Title, Year, Length, Type, Studio), starsIn(Title, Year, Name).
  q(Name) ← r2(Title, Year, Length, Type, Studio, Name).

  As in RA case, we could express this query using just one rule, as follows:
  q(Name) ← movie(Title, Year, Length, Type, Studio), Length >= 100,
  starsIn(Title, Year, Name).

Expressive Power of Datalog

- Relational algebra = Nonrecursive Datalog+ negation
- Datalog can express SQL **SELECT-FROM-WHERE** statements that do **not** use aggregation and/or grouping
- The SQL-99 standard supports recursion but it is not part of the "core" SQL-99 standard that every DBMS should support
- Some DBMS implementations, e.g. DB2, support *linear recursion*

Example

trainSchedule(From,To)

- Datalog:
  - fromMontreal(C) ← trainSchedule('Montreal', C).
  - fromMontreal(TC) ← fromMontreal(C), trainSchedule(C,TC).

- Find every cities that can be reached from Montreal by train

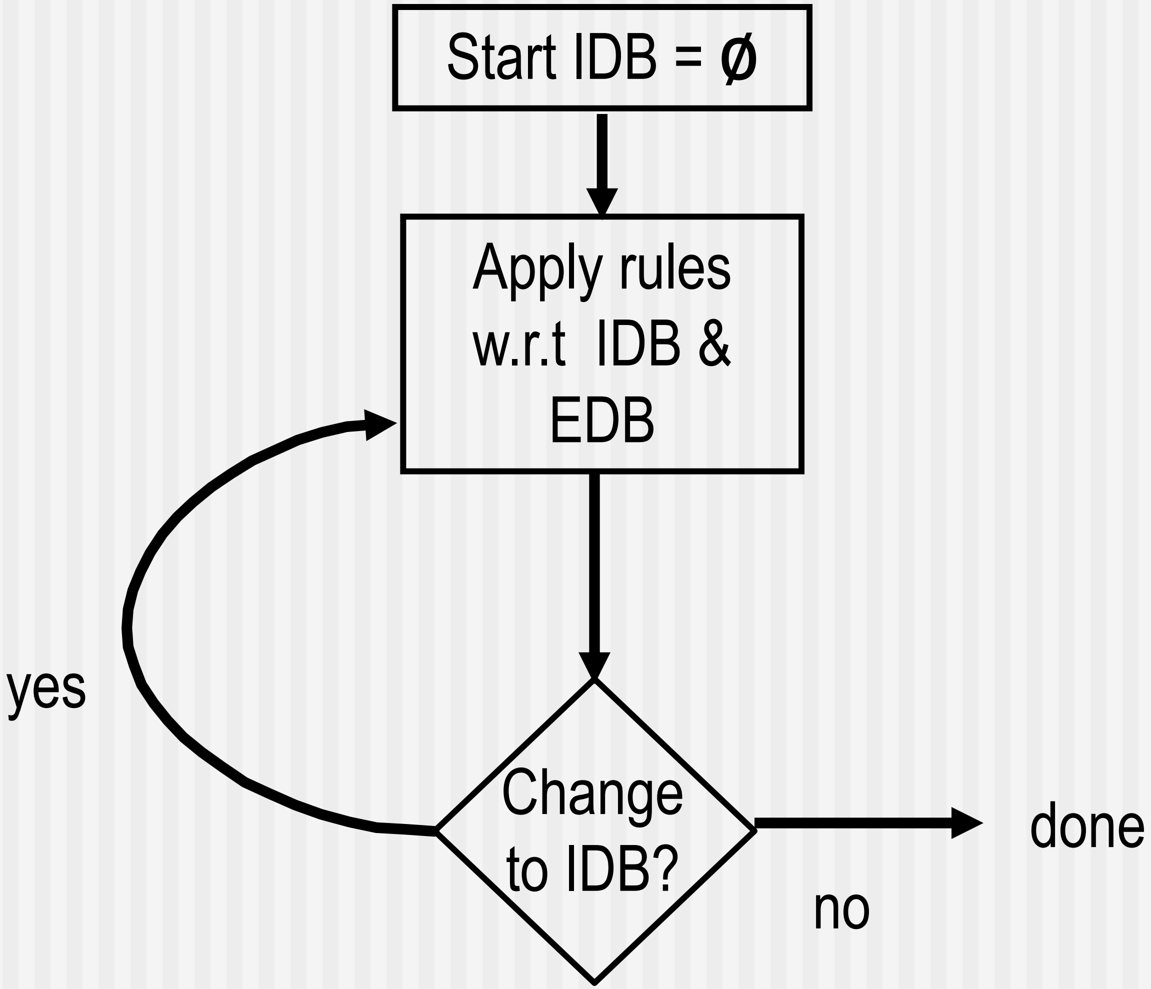
trainSchedule:

From	To
Toronto	Calgary
NYC	Boston
NYC	Albany
Chicago	Detroit
Montreal	Toronto
Montreal	NYC
Boston	Quebec City

fromMontreal:

Toronto
NYC
Calgary
Boston
Albany
Quebec City

Evaluation of Recursive Rules



Example

Relation Schema: **sequelOf**(Movie,Sequel)

Instance:

Movie	Sequel
Star wars	Star wars II
Naked Gun	Naked Gun 2
Naked Gun 2	Naked Gun 2 ½
Star wars II	Star wars III
Naked Gun 2 ½	Naked Gun 3

For a given movie, find all the follow-up movies, i.e., a sequel, a sequel of a sequel, and so on

Example

Relation Schema: **sequelOf**(Movie, Sequel)

Instance:

Movie	Sequel
Star wars	Star wars II
Naked Gun	Naked Gun 2
Naked Gun 2	Naked Gun 2 ½
Star wars II	Star wars III
Naked Gun 2 ½	Naked Gun 3

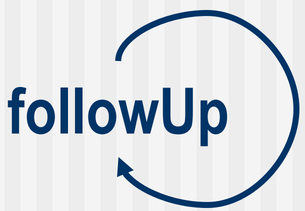
followUp(X, Y) $\leftarrow$ sequelOf(X,Y).
followUp(X, Y) $\leftarrow$ sequelOf(X,Z), followUp(Z,Y).

Recursion

- Let P be any datalog program
- We say an IDB predicate **r** in P *depends on* predicate s if there is a rule in P with r as the head and s as a subgoal in the rule body
- Construct the (dependency) graph of P:
  - **Nodes** -- **IDB** predicates in **P**
  - **Arcs** -- an arc from node **r** to **s** if **r** depends on **s**
  - Label the arc with '¬' for negated subgoals
- P is recursive iff its dependency graph has a cycle

Example

followUp(X, Y) $\leftarrow$ sequelOf(X,Y).
followUp(X, Y) $\leftarrow$ sequelOf(X,Z), followUp(Z, Y).



Safety

- It is possible to write a rule that makes “no sense”.
- Example of such rules:
  - $s(X) \leftarrow r(Y)$.
  - $s(X) \leftarrow$ **NOT** $r(X)$.
  - $s(X) \leftarrow r(Y), X < Y$.
- In each of these rules, the IDB relation **s** (output relation) could be infinite, even if (the input) relation ***r*** is finite
- Such rules are said to be not ***SAFE***

Safety

- For a rule to be safe, the following conditions must hold:
  - If a variable **X** appears in the rule head, then **X** must appear in an “ordinary” predicate in the body or be equal to such a variable (directly or indirectly), e.g., **X=Y**, and **Y** appears in an ordinary predicate in the rule body.

Recall: the predicates could be ordinary or built-in.