Deductive Databases and Datalog

Advanced Topics in Databases

Datalog

- The basic idea of Datalog is to use Horn clauses -- “if-then” rules – as a relational database query language.
- Relations are represented by predicates, e.g. Climbers, Climbs and Routes are interpreted as predicates with fixed arity.
- Arguments have a positional interpretation; e.g.

  Climbers(X,”Bridget”, “EXP”,”33”).
- The arguments can be constants (e.g. “Bridget”, “EXP”, and “33”) or variables (e.g. X).
- We will use uppercase letters for variables and lowercase letters for constants.

Truth values

- A predicate is ground if all of its arguments are constant. Ground predicates are **true**, which represents the fact that the "tuple" belongs to the relationship.

Climbers	CId	Cname	Skill	Age
	123	edmund	exp	80
	214	arnold	beg	25
	313	bridget	exp	33
	212	james	med	27

Climbers(313,bridget, exp,33) is true.
Climbers(518,jeremy,exp,17) is false.

- Predicates can also be negated:

NOT Climbers(518,jeremy,exp,17) is true.

Arithmetic predicates

- We will represent some arithmetic conditions and we will use the following built-in predicates <,<=,>,>=,=,<>.
- For instance, <(5,7) is true, but <(7,5) id false.
- Note that unlike “relational” predicates, arithmetic predicates are infinite!

Datalog rules

- A rule is of the form p←q where p is a relational predicate called **head** and q is a conjunction of predicates (subgoals) called **body**.
- Example:

$$\overset{\text{head}}{\text{EXPClimbers(I,N,A)}} \leftarrow \overbrace{\text{Climbers(I,N,S,A) AND S=exp}}^{\text{body}}$$

- When a variable is not used it can be replaced by an "_" (anonymous variable).

EXPClimbers(N)←Climbers(_,N,exp,_)

Some examples

- The names of the climbers with more than 32 years old:

OLD(N) ← Climbers(I,N,S,A) AND A>32

- The names of the climbers that climbed route 1:

Route1(N)←Climbers(I,N,_,_) AND Climbs(I, 1,_,_)

- The names of climbers under 40 who have climbed a route with grade greater than 5:

Rating5(N) ← Climbers(I,N,_,A) AND Climbs(I, R,_,_)
AND Routes(R,_,_,Ra,_) AND Ra>5 AND A<40

- Note the positional interpretation of attributes.

Safe rules

- A rule is safe if every variable occurs at least once in a positive relational predicate in the body.
- Some unsafe rules:

Likes(X,Y)←Starved(X)
Lazy(X) ←NOT Climbers(_,X,_,_)

- Some safe rules:

Likes(X,Y)←Starved(X) AND Food(Y)
Lazy(X) ←Person(X) AND NOT Climbers(_,X,_,_)

(and all the others we have seen so far)

Datalog query

- A query is a set of one or more rules. An empty body rule is a fact (a relational positive ground predicate).

```
Student (123,j.smith,compsci)
Student(456,k.tappet,french)
Offers(cookery,baking)
Offers(compsci,compilers)
Enroll(123,baking)
Enroll(012,compilers)

InterestedIn(X,S) ←Student(X,Y,S)
InterestedIn(X,S) ← Enroll(X,Z) AND Offers(S,Z)
```

The same query in relational algebra

- The previous query corresponds to the following expression from relational algebra:

$$\pi_{[1][3]}\text{Student} \cup \pi_{[1][4]}(\sigma_{[2]=[3]}(\text{Enroll} \times \text{Offers}))$$

Intentional versus extensional predicates

- Extensional predicates are those whose relations are stored in the database;
- Intentional predicates are those that are calculated applying one or more rules.
    - Student, Offers, and Enroll are extensional
    - InterestedIn is intentional
- Extensional predicates can never appear in the head of a rule.

Another example

```
Parent(mary,jane)
Parent(jane,fred)
Parent(ed,bob)
Parent(bob,fred)
Parent(fred,jill)

Ancestor(X,Y)←Parent(X,Y)
Ancestor(X,Y)←Parent(X,Z) AND Ancestor(Z,Y)
```

- Can we translate this Ancestor predicate to relational algebra?
- What would you expect the output of this query to be?
- Which predicates are extensional and which are intentional?

Meaning of Datalog rules

- Consider all possible assignments of values to variables. For each assignment that makes all subgoals true, the tuple corresponding to the head is true and added to the result.
- Example: X = 012, Z = compilers, S = compsci

```
Offers(compsci,compilers)
Enroll(012,compilers)
InterestedIn(X,S) ← Enroll(X,Z) AND Offers(S,Z)
```

thus ***InterestedIn(012, compsci)*** is added to the result.

Another way of defining meaning...

- The "attribution" method of defining meaning considers "meaningless" assignments to variables.
- For example: X = compilers, Z = 012, S = f.dunham

$$\text{InterestedIn}(X,S) \leftarrow \text{Enroll}(X,Z) \text{ AND } \text{Offers}(Z,S)$$

- Another method is to consider only the sets of tuples in each non-negated relational subgoal, and look just to assignments to "consistent" variables. If all subgoals are true (negated and arithmetic too), then the tuple in the head is added to the result.
- This suggests an implementation using AR!

An interesting query written “incorrectly”

- It is easy to write queries that do not express our intention. E. g.

$$\text{Single}(X) \leftarrow \text{Person}(X) \text{ AND NOT Married}(X,Y)$$

- What is the meaning of this query?
- If the goal was to obtain all married people, how should we write this query?
- This query is also not safe.

AR versus Datalog

- The Ancestor example is called recursive because the definition of ancestor depends on itself (directly).
- This cannot be simulated in AR, and we have to add a fixed-point operator to the algebra to simulate it.
- If subgoals cannot be negated, we cannot emulate set difference in Datalog.
- However, if subgoals can be denied we can simulate any AR expression in Datalog.

Simulation of AR in Datalog

- Intersection: simulate by a rule with each relation as a subgoal. E.g. remember Climbers and Hikers defined earlier.

$$\text{Climbers} \cap \text{Hikers}$$

would be written as:

CandH(I,N,S,A)←Climbers(I,N,S,A) AND Hikers(I,N,S,A)

- Difference: simulate by a rule with each relation as a subgoal, with the second subgoal negated. Thus Climbers – Hikers would be written as:

CnotH(I,N,S,A)←Climbers(I,N,S,A) AND NOT Hikers(I,N,S,A)

Simulation of AR in Datalog (cont.)

- Union: simulate by two rules, each with a single subgoal consisting of one of the relations. Thus:

$$\text{Climbers} \cup \text{Hikers}$$

would be written as:

$$\text{CorH}(I,N,S,A) \leftarrow \text{Climbers}(I,N,S,A)$$
$$\text{CorH}(I,N,S,A) \leftarrow \text{Hikers}(I,N,S,A)$$

- Projection: simulate by a rule, whose head uses the variables that correspond to the attributes we want to project. Thus,

$$\pi_{Name,Age}\text{Climbers}$$

would be written as:

$$\text{Result}(N,A) \leftarrow \text{Climbers}(\_,N,\_,A)$$

Simulation of selection

- If the condition is a conjunction of arithmetic atoms, it becomes easy: join each atom as a sub-goal. Thus:

$$\sigma_{Name="Bridget" \wedge Age>33} \text{Climbers}$$

would be written as:

Result(I,N,S,A)←Climbers(I,N,S,A) AND N= bridget
AND Age>30

- Or can be simulated using the union; recall the equivalence:

$$\sigma_{C \vee D} R \equiv \sigma_C R \bigcup \sigma_D R$$

Simulation of selection (cont.)

- Now, recall that any expression involving AND, OR and NOT can be put in the normal disjunctive form: an OR of conjunctions, each of which is an AND of comparisons.

$$\sigma_{\neg(Age \le 30 \vee Name = "Bridget") \vee Skill = EXP} \text{Climbers} \approx$$

$$\sigma_{(\neg Age \le 30 \wedge \neg Name = "Bridget") \vee Skill = EXP} \text{Climbers} \approx$$

$$\sigma_{(Age > 30 \wedge Name \ne "Bridget") \vee Skill = EXP} \text{Climbers} \approx$$

$$\sigma_{Age > 30 \wedge Name \ne "Bridget"} \text{Climbers} \cup \sigma_{Skill = EXP} \text{Climbers}$$

Simulation of cartesian product and join

- The product of two relations is expressed by a single rule with both relations as subgoals; all relations variables appear in the head:

$$R(A,B) \leftarrow S(A) \text{ AND } T(B)$$

- A join is a selection on equality followed by a projection on a product;

  Equality can be expressed by reusing the same variables:

$$R(A,B,C) \leftarrow S(A,B) \text{ AND } T(D,C) \text{ AND } B=D$$ or

$$R(A,B,C) \leftarrow S(A,B) \text{ AND } T(B,C)$$

Simulation of multiple operations of AR

- Creation of the “operators tree”:

$$\pi_{Name,RName,Duration}\,\sigma_{Name="Jeremy"}(\text{Climbers}\infty\text{Climbs})$$

$$\begin{tikzcd}
 & \pi_{Name,RName,Duration} \arrow[d] & \\
 & \sigma_{Name="Jeremy"} \arrow[d] & \\
 & \infty \arrow[dl] \arrow[dr] & \\
\textbf{Climbers} & & \textbf{Climbs}
\end{tikzcd}$$

- Create an intentional predicate for each internal node of the tree and write the corresponding rule.
- The intentional predicate corresponding to the root of the tree is the result.

Datalog: dependency graph

- Leaves correspond to relational predicates;
- There is a branch from A to B if B appears as a subgoal (positive or negative) in a rule with head A. The branch is annotated with “-” for negated subgoals.

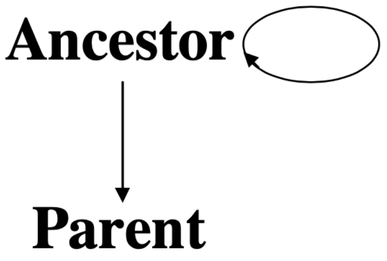
InterestedIn(X,S) $\leftarrow$Student(X,Y,S) InterestedIn(X,S) $\leftarrow$
Enroll(X,Z) AND Offers(Z,S)

$$\begin{tikzcd}
 & \textbf{InterestedIn} \arrow[dl] \arrow[d] \arrow[dr] & \\
\textbf{Student} & \textbf{Enroll} & \textbf{Offers}
\end{tikzcd}$$

Cyclic dependency graphs

- A cyclic dependency graph indicates a recursive query.
- Recall the ancestor example:

$$\text{Ancestor}(X,Y) \leftarrow \text{Parent}(X,Y)$$
$$\text{Ancestor}(X,Y) \leftarrow \text{Parent}(X,Z) \text{ AND } \text{Ancestor}(Z,Y)$$



Datalog evaluation in AR

- Assume non-recursive Datalog programs for now and without negated predicates.
- The evaluation of intentional database predicates proceeds in a “bottom-up”manner starting at the leaves of the graph of dependencies for subgoals to be fully evaluated when they are used. (notice that leaves correspond to extensional databases predicates!)

Datalog evaluation in AR (cont.)

- Procedure for evaluating a rule (without negation):
  - Take the product of relational (non-arithmetic) subgoals;
  - Form a selection of the product with a condition that equates the arguments with the same variable and all arithmetic predicates;
  - Project on the variables that appear in the head;
- For each intentional predicate R, take the union of R-headed rule expressions.

Examples

Result(N)←Climbers(I,N,_,A) AND A<40 AND
Climbs(I,R,_,_) AND Routes(R,_,_,Ra,_) AND Ra>5

$$\pi_{[2]}(\sigma_{[1]=[5] \wedge [6]=[9] \wedge [4]<40 \wedge [12]>5}(\text{Climbers} \times \text{Climbs} \times \text{Routes}))$$

InterestedIn(X,S) ←Student(X,Y,S)
InterestedIn(X,S) ← Enroll(X,Z) AND Offers(Z,S)

$$\pi_{[1][3]}\text{Student} \cup \pi_{[1][4]}(\sigma_{[2]=[3]}(\text{Enroll} \times \text{Offers}))$$

NotSingle(X) ←Married(X,Y)
NotSingle(X) ←Married(Y,X)

$$\pi_{[1]}\text{Married} \cup \pi_{[2]}\text{Married}$$

Treatment of negated subgoals

- Suppose NOT R(X,Y,Z) appears as a negated subgoal in a query.
- Let DOM={any symbol that occurs in the rule} U {any symbol that appears in any relational instance that appears in a subgoal of the rule}.
- It is sufficient to "evaluate" NOT R(X,Y,Z) as (DOM X DOM X DOM)-R
- However, there are problems when negation is combined with recursion.

Example

- The correct version of the example that would get all single people in the database is:

$$\begin{array}{l}\text{NotSingle}(X) \leftarrow \text{Married}(X,Y) \\ \text{NotSingle}(X) \leftarrow \text{Married}(Y,X) \\ \text{Single}(X) \leftarrow \text{Person}(X) \text{ AND NOT NotSingle}(X)\end{array}$$

- How do we evaluate this query?
- Let DOM={any symbol in Person}U{any symbol in NotSingle}

$$\pi_{[1]}(\sigma_{[1]=[2]}(\text{Person} \times (\text{DOM} - \text{NotSingle})))$$

Recursive queries (no negation) – “Naïve” evaluation

- Let R, S, ... be intentional predicates occurring in a single cycle of the dependency graph.

```
$R=\emptyset$, $S=\emptyset$
while there is a change to R, S,… do
    $R=R \cup \{\text{evaluation of } R\}$
    $S=S \cup \{\text{evaluation of } S\}$
```

Example

Parent(mary,jane)	Ancestor(X,Y)←Parent(X,Y)
Parent(jane,fred)	
Parent(ed,bob)	Ancestor(X,Y)←Parent(X,Z)
Parent(bob,fred)	AND Ancestor(Z,Y)
Parent(fred,jill)	

- Evaluation of Ancestor:

$$\text{Parent} \cup (\pi_{[1],[4]}\sigma_{[2]=[3]}(\text{Parent} \times \text{Ancestor}))$$

1. Ancestor = ø
2. Ancestor = {(mary,jane),(jane,fred),(ed,bob),(bob,fred),
   (fred,jill)}

Example (cont.)

$$\text{Parent} \cup (\pi_{[1],[4]} \sigma_{[2]=[3]} (\text{Parent} \times \text{Ancestor}))$$

3. Ancestor = {(mary,jane),(jane,fred),(ed,bob),(bob,fred),
   (fred,jill), (mary, fred),(jane,jill),(ed.fred)
   (bob,jill)}
4. Ancestor = {(mary,jane),(jane,fred),(ed,bob),(bob,fred),
   (fred,jill), (mary, fred),(jane,jill),(ed,fred)
   (bob,jill), (mary,jill), (ed,jill)}