

Logical Query Languages

Motivation:

1. Logical rules extend more naturally to recursive queries than does relational algebra.
 - ◆ Used in SQL recursion.
2. Logical rules form the basis for many information-integration systems and applications.

1

Datalog Example

```
Likes(drinker, beer)
Sells(bar, beer, price)
Frequents(drinker, bar)

Happy(d) <-
  Frequents(d,bar) AND
  Likes(d,beer) AND
  Sells(bar,beer,p)
```

- Above is a *rule*.
- Left side = *head*.
- Right side = *body* = AND of *subgoals*.
- Head and subgoals are *atoms*.
 - ◆ Atom = *predicate* and arguments.
 - ◆ Predicate = relation name or arithmetic predicate, e.g. <.
 - ◆ Arguments are variables or constants.
- Subgoals (not head) may optionally be negated by NOT.

2

Meaning of Rules

Head is true of its arguments if there exist values for *local* variables (those in body, not in head) that make all of the subgoals true.

- If no negation or arithmetic comparisons, just natural join the subgoals and project onto the head variables.

Example

Above rule equivalent to $\text{Happy}(d) = \pi_{\text{drinker}}(\text{Frequents} \bowtie \text{Likes} \bowtie \text{Sells})$

3

Evaluation of Rules

Two, dual, approaches:

1. *Variable-based*: Consider all possible assignments of values to variables. If all subgoals are true, add the head to the result relation.
2. *Tuple-based*: Consider all assignments of tuples to subgoals that make each subgoal true. If the variables are assigned consistent values, add the head to the result.

Example: Variable-Based Assignment

```
S(x,y) <- R(x,z) AND R(z,y)
AND NOT R(x,y)
```

R =

A	B
1	2
2	3

4

- Only assignments that make first subgoal true:
 1. $x \rightarrow 1, z \rightarrow 2$.
 2. $x \rightarrow 2, z \rightarrow 3$.
- In case (1), $y \rightarrow 3$ makes second subgoal true. Since (1,3) is *not* in R , the third subgoal is also true.
 - ◆ Thus, add $(x,y) = (1,3)$ to relation S .
- In case (2), no value of y makes the second subgoal true. Thus, $S =$

A	B
1	3

5

Example: Tuple-Based Assignment

Trick: start with the positive (not negated), relational (not arithmetic) subgoals only.

$$S(x,y) \leftarrow R(x,z) \text{ AND } R(z,y) \\ \text{AND NOT } R(x,y)$$

$R =$

A	B
1	2
2	3

- Four assignments of tuples to subgoals:

$R(x,z)$	$R(z,y)$
(1,2)	(1,2)
(1,2)	(2,3)
(2,3)	(1,2)
(2,3)	(2,3)

- Only the second gives a consistent value to z .
- That assignment also makes NOT $R(x,y)$ true.
- Thus, (1,3) is the only tuple for the head.

6

Safety

A rule can make no sense if variables appear in funny ways.

Examples

- $S(x) \leftarrow R(y)$
- $S(x) \leftarrow \text{NOT } R(x)$
- $S(x) \leftarrow R(y) \text{ AND } x < y$

In each of these cases, the result is infinite, even if the relation R is finite.

- To make sense as a database operation, we need to require three things of a variable x (= definition of *safety*). If x appears in either
 1. The head,
 2. A negated subgoal, or
 3. An arithmetic comparison,
 then x must also appear in a nonnegated, "ordinary" (relational) subgoal of the body.
- We insist that rules be safe, henceforth.

7

Datalog Programs

- A collection of rules is a *Datalog program*.
- Predicates/relations divide into two classes:
 - ◆ EDB = *extensional database* = relation stored in DB.
 - ◆ IDB = *intensional database* = relation defined by one or more rules.
- A predicate must be IDB or EDB, not both.
 - ◆ Thus, an IDB predicate can appear in the body or head of a rule; EDB only in the body.

8

Example

Convert the following SQL (Find the manufacturers of the beers Joe sells):

```
Beers(name, manf)
Sells(bar, beer, price)

SELECT manf
FROM Beers
WHERE name IN (
  SELECT beer
  FROM Sells
  WHERE bar = 'Joe''s Bar'
);
```

to a Datalog program.

```
JoeSells(b) <-
  Sells('Joe''s Bar', b, p)
Answer(m) <-
  JoeSells(b) AND Beers(b,m)
```

- Note: Beers, Sells = EDB; JoeSells, Answer = IDB.

Expressive Power of Datalog

- Nonrecursive Datalog = (classical) relational algebra.
 - ◆ See discussion in text.
- Datalog simulates SQL select-from-where without aggregation and grouping.
- Recursive Datalog expresses queries that cannot be expressed in SQL.
- But none of these languages have full expressive power (*Turing completeness*).

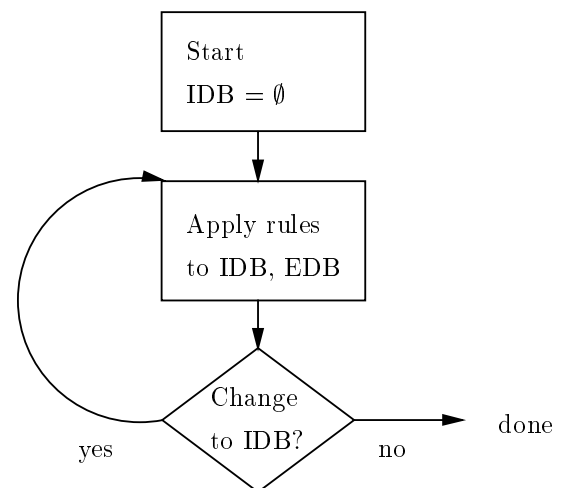
Recursion

- IDB predicate P depends on predicate Q if there is a rule with P in the head and Q in a subgoal.
- Draw a graph: nodes = IDB predicates, arc $P \rightarrow Q$ means P depends on Q .
- Cycles iff recursive.

Recursive Example

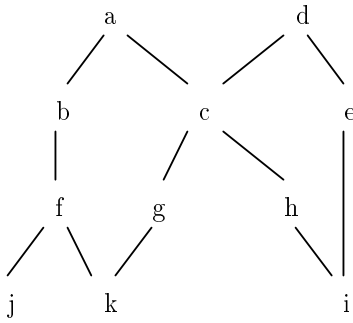
```
Sib(x,y) <- Par(x,p) AND Par(y,p)
  AND x <> y
Cousin(x,y) <- Sib(x,y)
Cousin(x,y) <- Par(x,yp)
  AND Par(y,yp)
  AND Cousin(xp,yp)
```

Iterative Fixed-Point Evaluates Recursive Rules



Example

EDB Par =



- Note, because of symmetry, **Sib** and **Cousin** facts appear in pairs, so we shall mention only (x, y) when both (x, y) and (y, x) are meant.

13

	Sib	Cousin
Initial	\emptyset	\emptyset
Round 1	$(b, c), (c, e)$	\emptyset
add:	$(g, h), (j, k)$	
Round 2		$(b, c), (c, e)$
add:		$(g, h), (j, k)$
Round 3		$(f, g), (f, h)$
add:		$(g, i), (h, i)$
		(i, k)
Round 4		(k, k)
add:		(i, j)

14

Stratified Negation

- Negation wrapped inside a recursion makes no sense.
- Even when negation and recursion are separated, there can be ambiguity about what the rules mean, and some one meaning must be selected.
- *Stratified negation* is an additional restraint on recursive rules (like safety) that solves both problems:
 1. It rules out negation wrapped in recursion.
 2. When negation is separate from recursion, it yields the intuitively correct meaning of rules (the *stratified model*).

15

Problem with Recursive Negation

Consider:

$$P(x) \leftarrow Q(x) \text{ AND NOT } P(x)$$

- $Q = \text{EDB} = \{1, 2\}$.
- Compute IDB P iteratively?
 - ◆ Initially, $P = \emptyset$.
 - ◆ Round 1: $P = \{1, 2\}$.
 - ◆ Round 2: $P = \emptyset$, etc., etc.

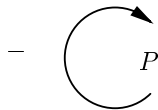
16

Strata

Intuitively: stratum of an IDB predicate = maximum number of negations you can pass through on the way to an EDB predicate.

- Must not be ∞ in “stratified” rules.
- Define *stratum graph*:
 - ◆ Nodes = IDB predicates.
 - ◆ Arc $P \rightarrow Q$ if Q appears in the body of a rule with head P .
 - ◆ Label that arc $-$ if Q is in a negated subgoal.

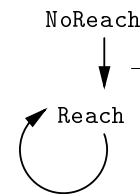
Example

$$P(x) \leftarrow Q(x) \text{ AND NOT } P(x)$$


17

Example

Which target nodes cannot be reached from any source node?

$$\begin{aligned} \text{Reach}(x) &\leftarrow \text{Source}(x) \\ \text{Reach}(x) &\leftarrow \text{Reach}(y) \text{ AND Arc}(y,x) \\ \text{NoReach}(x) &\leftarrow \text{Target}(x) \\ &\text{AND NOT Reach}(x) \end{aligned}$$


18

Computing Strata

Stratum of an IDB predicate A = maximum number of $-$ arcs on any path from A in the stratum graph.

Examples

- For first example, stratum of P is ∞ .
- For second example, stratum of Reach is 0; stratum of NoReach is 1.

Stratified Negation

A Datalog program is *stratified* if every IDB predicate has a finite stratum.

Stratified Model

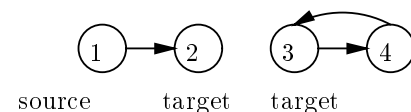
If a Datalog program is stratified, we can compute the relations for the IDB predicates lowest-stratum-first.

19

Example

$$\begin{aligned} \text{Reach}(x) &\leftarrow \text{Source}(x) \\ \text{Reach}(x) &\leftarrow \text{Reach}(y) \text{ AND Arc}(y,x) \\ \text{NoReach}(x) &\leftarrow \text{Target}(x) \\ &\text{AND NOT Reach}(x) \end{aligned}$$

- EDB:
 - ◆ $\text{Source} = \{1\}$.
 - ◆ $\text{Arc} = \{(1,2), (3,4), (4,3)\}$.
 - ◆ $\text{Target} = \{2,3\}$.



- First compute $\text{Reach} = \{1, 2\}$ (stratum 0).
- Next compute $\text{NoReach} = \{3\}$.

20

Is the Stratified Solution “Obvious”?

Not really.

- There is another model that makes the rules true no matter what values we substitute for the variables.
 - ◆ $\text{Reach} = \{1, 2, 3, 4\}$.
 - ◆ $\text{NoReach} = \emptyset$.
- Remember: the only way to make a Datalog rule false is to find values for the variables that make the body true and the head false.
 - ◆ For this model, the heads of the rules for Reach are true for all values, and in the rule for NoReach the subgoal $\text{NOT Reach}(x)$ assures that the body cannot be true.