Introduction to Data Management CSE 344

Lecture 12: Relational Calculus

Announcements

- WQ4 due on Thursday
- Homework 3 due on Friday night
- Midterm: Monday, November 4^{th}, in class

- For the material in the last three lectures: optional reading *Query Language Primer*, posted on the website

Friend(name1, name2)
Enemy(name1, name2)

Review: Datalog

Find Joe's friends, and Joe's friends of friends.

```
A(x) :- Friend('Joe', x)
A(x) :- Friend('Joe', z), Friend(z, x)
```

Friend(name1, name2)
Enemy(name1, name2)

Review: Datalog+negation

Find all of Joe's friends who do not have any friends except for Joe:

```
NonAns(x) :- Friend(x,y), y != ‘Joe’
A(x) :- Friend('Joe',x), NOT NonAns(x)
```

Person(name)
Friend(name1, name2)
Enemy(name1, name2)

Review: Datalog+negation

Find all people such that all their enemies' enemies are their friends

- Assume that if someone doesn't have any enemies nor friends, we also want them in the answer

```
NonAns(x) :- Enemy(x,y),Enemy(y,z), NOT Friend(x,z)
A(x) :- Person(x), NOT NonAns(x)
```

Person(name)
Friend(name1, name2)
Enemy(name1, name2)

Review: Datalog+negation

Find all persons x having some friend all of whose enemies are x's enemies.

```
NonAns(x) :- Friend(x,y), Enemy(y,z), NOT Enemy(x,z)
A(x) :- Person(x), NOT NonAns(x)
```

Datalog Summary

- EDB and IDB
- Datalog program = set of rules
- Datalog is recursive
- Pure datalog does not have negation;
  if we want negation we say “datalog+negation”
- Multiple atoms in a rule mean join (or intersection)
- Multiple rules with same head mean union
- All variables in the body are existentially quantified
- If we need universal quantifiers, we use DeMorgan’s laws and negation

Relational Calculus

- Aka *predicate calculus* or *first order logic*

- TRC = Tuple RC
  – See book
- DRC = Domain RC = unnamed perspective
  – We study only this one
  – Also see: *Query Language Primer*

Relational Calculus

Relational predicate P is a formula given by this grammar:

$$P ::= \text{atom} \mid P \wedge P \mid P \vee P \mid P \Rightarrow P \mid \text{not}(P) \mid \forall x.P \mid \exists x.P$$

Query Q:

$$Q(x1, \ldots, xk) = P$$

Relational Calculus

Relational predicate P is a formula given by this grammar:

$$P ::= atom \mid P \wedge P \mid P \vee P \mid P \Rightarrow P \mid not(P) \mid \forall x.P \mid \exists x.P$$

Query Q:

$$Q(x1, \ldots, xk) = P$$

Example: find the first/last names of actors who acted in 1940

$$Q(f,l) = \exists x.\ \exists y.\ \exists z.\ (Actor(z,f,l) \wedge Casts(z,x) \wedge Movie(x,y,1940))$$

What does this query return ?

$$Q(f,l) = \exists z.\ (Actor(z,f,l) \wedge \forall x.(Casts(z,x) \Rightarrow \exists y.Movie(x,y,1940)))$$

Likes(drinker, beer)
Frequents(drinker, bar)
Serves(bar, beer)

Important Observation

Find all bars that serve all beers that Fred likes

$$A(x) = \forall y.\ Likes("Fred", y) \Rightarrow Serves(x,y)$$

- Note: P => Q (read P implies Q) is the same as (not P) OR Q
  In this query: If Fred likes a beer the bar must serve it (P => Q)
  In other words: Either Fred does not like the beer (not P) OR the bar serves that beer (Q).

$$A(x) = \forall y.\ not(Likes("Fred", y))\ OR\ Serves(x,y)$$

Likes(drinker, beer)
Frequents(drinker, bar)
Serves(bar, beer)

More Examples

Average Joe

Find drinkers that frequent some bar that serves some beer they like.

Likes(drinker, beer)
Frequents(drinker, bar)
Serves(bar, beer)

More Examples

Average Joe

Find drinkers that frequent <u>some</u> bar that serves <u>some</u> beer they like.

$$\boxed{Q(x) = \exists y.\ \exists z.\ \text{Frequents}(x, y) \land \text{Serves}(y,z) \land \text{Likes}(x,z)}$$

Likes(drinker, beer)
Frequents(drinker, bar)
Serves(bar, beer)

More Examples

Average Joe

Find drinkers that frequent some bar that serves some beer they like.

$$Q(x) = \exists y.\ \exists z.\ \text{Frequents}(x, y) \land \text{Serves}(y,z) \land \text{Likes}(x,z)$$

Prudent Peter

Find drinkers that frequent only bars that serves some beer they like.

Likes(drinker, beer)
Frequents(drinker, bar)
Serves(bar, beer)

More Examples

Average Joe

Find drinkers that frequent <u>some</u> bar that serves <u>some</u> beer they like.

$$Q(x) = \exists y.\ \exists z.\ \mathrm{Frequents}(x, y) \wedge \mathrm{Serves}(y,z) \wedge \mathrm{Likes}(x,z)$$

Prudent Peter

Find drinkers that frequent <u>only</u> bars that serves <u>some</u> beer they like.

$$Q(x) = \forall y.\ \mathrm{Frequents}(x, y) \Rightarrow (\exists z.\ \mathrm{Serves}(y,z) \wedge \mathrm{Likes}(x,z))$$

Likes(drinker, beer)
Frequents(drinker, bar)
Serves(bar, beer)

More Examples

Average Joe

Find drinkers that frequent <u>some</u> bar that serves <u>some</u> beer they like.

$$Q(x) = \exists y.\ \exists z.\ \text{Frequents}(x, y) \wedge \text{Serves}(y,z) \wedge \text{Likes}(x,z)$$

Prudent Peter

Find drinkers that frequent <u>only</u> bars that serves <u>some</u> beer they like.

$$Q(x) = \forall y.\ \text{Frequents}(x, y) \Rightarrow (\exists z.\ \text{Serves}(y,z) \wedge \text{Likes}(x,z))$$

Cautious Carl

Find drinkers that frequent <u>some</u> bar that serves <u>only</u> beers they like.

Likes(drinker, beer)
Frequents(drinker, bar)
Serves(bar, beer)

More Examples

Average Joe

Find drinkers that frequent <u>some</u> bar that serves <u>some</u> beer they like.

$$Q(x) = \exists y.\ \exists z.\ Frequents(x, y) \wedge Serves(y,z) \wedge Likes(x,z)$$

Prudent Peter

Find drinkers that frequent <u>only</u> bars that serves <u>some</u> beer they like.

$$Q(x) = \forall y.\ Frequents(x, y) \Rightarrow (\exists z.\ Serves(y,z) \wedge Likes(x,z))$$

Cautious Carl

Find drinkers that frequent <u>some</u> bar that serves <u>only</u> beers they like.

$$Q(x) = \exists y.\ Frequents(x, y) \wedge \forall z.(Serves(y,z) \Rightarrow Likes(x,z))$$

Likes(drinker, beer)
Frequents(drinker, bar)
Serves(bar, beer)

More Examples

Average Joe

Find drinkers that frequent some bar that serves some beer they like.

$$Q(x) = \exists y.\ \exists z.\ \text{Frequents}(x, y) \wedge \text{Serves}(y,z) \wedge \text{Likes}(x,z)$$

Prudent Peter

Find drinkers that frequent only bars that serves some beer they like.

$$Q(x) = \forall y.\ \text{Frequents}(x, y) \Rightarrow (\exists z.\ \text{Serves}(y,z) \wedge \text{Likes}(x,z))$$

Cautious Carl

Find drinkers that frequent some bar that serves only beers they like.

$$Q(x) = \exists y.\ \text{Frequents}(x, y) \wedge \forall z.(\text{Serves}(y,z) \Rightarrow \text{Likes}(x,z))$$

Paranoid Paul

Find drinkers that frequent only bars that serves only beer they like.

Likes(drinker, beer)
Frequents(drinker, bar)
Serves(bar, beer)

More Examples

Average Joe

Find drinkers that frequent <u>some</u> bar that serves <u>some</u> beer they like.

$$Q(x) = \exists y.\ \exists z.\ \text{Frequents}(x, y) \land \text{Serves}(y,z) \land \text{Likes}(x,z)$$

Prudent Peter

Find drinkers that frequent <u>only</u> bars that serves <u>some</u> beer they like.

$$Q(x) = \forall y.\ \text{Frequents}(x, y) \Rightarrow (\exists z.\ \text{Serves}(y,z) \land \text{Likes}(x,z))$$

Cautious Carl

Find drinkers that frequent <u>some</u> bar that serves <u>only</u> beers they like.

$$Q(x) = \exists y.\ \text{Frequents}(x, y) \land \forall z.(\text{Serves}(y,z) \Rightarrow \text{Likes}(x,z))$$

Paranoid Paul

Find drinkers that frequent <u>only</u> bars that serves <u>only</u> beer they like.

$$Q(x) = \forall y.\ \text{Frequents}(x, y) \Rightarrow \forall z.(\text{Serves}(y,z) \Rightarrow \text{Likes}(x,z))$$

Likes(drinker, beer)
Frequents(drinker, bar)
Serves(bar, beer)

Domain Independent Relational Calculus

- As in datalog, one can write "unsafe" RC queries; they are also called ***domain dependent***

```
A(x) = not Likes("Fred", x)
A(x,y) = Likes("Fred", x) OR Serves("Bar", y)
A(x) = ∀y. Serves(x,y)
```

- Lesson: make sure your RC queries are domain independent

Relational Calculus

How to write a complex SQL query:

- Write it in RC
- Translate RC to datalog
- Translate datalog to SQL

Take shortcuts when you know what you're doing

Likes(drinker, beer)
Frequents(drinker, bar)
Serves(bar, beer)

From RC to Datalog$^{\neg}$ to SQL

Query: Find drinkers that like some beer so much that they frequent all bars that serve it

$$Q(x) = \exists y.\ Likes(x, y) \land \forall z.(Serves(z,y) \Rightarrow Frequents(x,z))$$

Likes(drinker, beer)
Frequents(drinker, bar)
Serves(bar, beer)

From RC to Datalog$^{\neg}$ to SQL

Query: Find drinkers that like some beer so much that they frequent all bars that serve it

$$Q(x) = \exists y.\ \text{Likes}(x, y) \wedge \forall z.(\text{Serves}(z,y) \Rightarrow \text{Frequents}(x,z))$$

$\forall x\ P(x)$ same as $\neg\exists x\ \neg P(x)$

Step 1: Replace $\forall$ with $\exists$ using de Morgan's Laws

$$Q(x) = \exists y.\ \text{Likes}(x, y) \wedge \neg\exists z.(\text{Serves}(z,y) \wedge \neg\text{Frequents}(x,z))$$

$\neg(\neg P \vee Q)$ same as $P \wedge \neg Q$

Likes(drinker, beer)
Frequents(drinker, bar)
Serves(bar, beer)

From RC to Datalog$^{\neg}$ to SQL

Query: Find drinkers that like some beer so much that they frequent all bars that serve it

$$Q(x) = \exists y.\ \text{Likes}(x, y) \wedge \forall z.(\text{Serves}(z,y) \Rightarrow \text{Frequents}(x,z))$$

$\forall x\ P(x)$ same as $\neg\exists x\ \neg P(x)$

Step 1: Replace ∀ with ∃ using de Morgan's Laws

$$Q(x) = \exists y.\ \text{Likes}(x, y) \wedge \neg\exists z.(\text{Serves}(z,y) \wedge \neg\text{Frequents}(x,z))$$

$\neg(\neg P \vee Q)$ same as $P \wedge \neg Q$

Step 2: Make all *subqueries* domain independent

$$Q(x) = \exists y.\ \text{Likes}(x, y) \wedge \neg\exists z.(\text{Likes}(x,y) \wedge \text{Serves}(z,y) \wedge \neg\text{Frequents}(x,z))$$

Likes(drinker, beer)
Frequents(drinker, bar)
Serves(bar, beer)

From RC to Datalog$^{\neg}$ to SQL

$$Q(x) = \exists y.\ Likes(x, y) \wedge \neg \underbrace{\exists z.(Likes(x,y) \wedge Serves(z,y) \wedge \neg Frequents(x,z))}_{H(x,y)}$$

Step 3: Create a datalog rule for each subexpression;
(shortcut: only for "important" subexpressions)

```
H(x,y)   :- Likes(x,y),Serves(z,y), not Frequents(x,z)
Q(x)     :- Likes(x,y), not H(x,y)
```

Likes(drinker, beer)
Frequents(drinker, bar)
Serves(bar, beer)

From RC to Datalog$^{\neg}$ to SQL

```
H(x,y)   :- Likes(x,y),Serves(z,y), not Frequents(x,z)
Q(x)     :- Likes(x,y), not H(x,y)
```

Step 4: Write it in SQL

```
SELECT DISTINCT L.drinker FROM Likes L
WHERE ……
```

Likes(drinker, beer)
Frequents(drinker, bar)
Serves(bar, beer)

From RC to Datalog$^{\neg}$ to SQL

```
H(x,y)   :- Likes(x,y),Serves(z,y), not Frequents(x,z)
Q(x)     :- Likes(x,y), not H(x,y)
```

Step 4: Write it in SQL

```
SELECT DISTINCT L.drinker FROM Likes L
WHERE not exists
   (SELECT * FROM Likes L2, Serves S
    WHERE … …)
```

Likes(drinker, beer)
Frequents(drinker, bar)
Serves(bar, beer)

From RC to Datalog$^{\neg}$ to SQL

```
H(x,y)   :- Likes(x,y),Serves(z,y), not Frequents(x,z)
Q(x)     :- Likes(x,y), not H(x,y)
```

Step 4: Write it in SQL

```
SELECT DISTINCT L.drinker FROM Likes L
WHERE not exists
  (SELECT * FROM Likes L2, Serves S
   WHERE L2.drinker=L.drinker and L2.beer=L.beer
         and L2.beer=S.beer
         and not exists (SELECT * FROM Frequents F
                              WHERE F.drinker=L2.drinker
                                 and F.bar=S.bar))
```

Likes(drinker, beer)
Frequents(drinker, bar)
Serves(bar, beer)

From RC to Datalog$^{\neg}$ to SQL

```
H(x,y)   :- ~~Likes(x,y),~~Serves(z,y), not Frequents(x,z)
Q(x)     :- Likes(x,y), not H(x,y)
```

Unsafe rule

Improve the SQL query by using an unsafe datalog rule

```
SELECT DISTINCT L.drinker FROM Likes L
WHERE not exists
  (SELECT * FROM Serves S
   WHERE L.beer=S.beer
         and not exists (SELECT * FROM Frequents F
                         WHERE F.drinker=L.drinker
                             and F.bar=S.bar))
```

Summary: all these formalisms are equivalent!

- We have seen these translations:
  - RA → datalog¬
  - RC → datalog¬
- Practice at home, or read *Query Language Primer:*
  - Nonrecursive datalog¬ → RA
  - RA → RC
- Summary:
  - RA, RC, and non-recursive datalog¬ can express the same class of queries, called Relational Queries