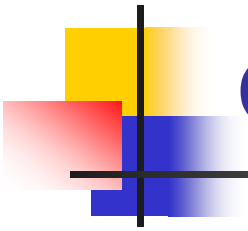
CS589 Principles of DB Systems
Fall 2008
Lecture 4c: Query Language Equivalence

Lois Delcambre
lmd@cs.pdx.edu
503 725-2405



Goal for this lecture

- Demonstrate how we can prove that one query language is more expressive than (what the book calls "contained in") another.
  - Introduce the way the proofs use mathematical induction
  - Walk through some of the proofs from the book
  - Summarize the results of QL equivalence

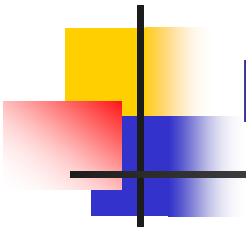
Equivalence of relational query languages

Two queries are equivalent if they return the same answer for any possible DB state.

One QL_2 is more expressive than QL_1 if we can prove that every query expressible in QL_1 can be expressed in QL_2. QL_1 and QL_2 are equivalent if you can prove "more expressive" or "contained in" in both directions.

The following four query languages are equivalent.

- Relational algebra
- Safe, non-recursive datalog programs with negation
- Allowed domain calculus (and allowed tuple calculus)



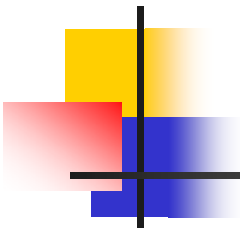
How do we prove it?

Complete the circle

1. Prove relational algebra is contained in safe, non-recursive Datalog with negation
2. Prove safe, non-recursive Datalog is contained in allowable domain calculus with negation
3. Prove allowable domain relational calculus is contained in relational algebra

Then we will know that all three languages are equivalent.

1. Prove relational algebra is contained in safe, non-recursive Datalog



We need to take an arbitrary relational algebra query and show how to construct an equivalent safe, non-recursive Datalog program.

How shall we frame the proof? How do we take an arbitrary relational algebra query expression?

By induction on the number of operators that appear in the query expression.

We need:

a (minimal) list of the operators in relational algebra.

a base case.

the inductive hypotheses.

1. Prove relational algebra is contained in safe, non-recursive Datalog (continued)

Minimal set of operators (with the full expressive power of the relational algebra):

$\cup$, -, π, $\bowtie$, ν

+By the way, how would you prove that this is, in fact, a minimal set of operators?)

Base case for the induction:

a relational algebra expression with zero operators.

that is, a query of the form: R

What is the equivalent Datalog program?

answer(x_1, x_2, …, x_n) :- R(x_1, x_2, …, x_n).

or just R(x_1, x_2, …, x_n).

The induction step

- Assume that the theorem is true for all relational algebra expression with q or fewer operators. Then consider a relational algebra expression with q+1 operators.
- What can that $(q+1)^{st}$ operator be?

  One of the operators in our minimal set.

UNION: if the query expression Q is $Q_1 \cup Q_2$ then the equivalent safe, non-recursive Datalog program is:

$Q(x_1, x_2, \ldots, x_n) :- Q_1(x_1, x_2, \ldots, x_n).$
$Q(x_1, x_2, \ldots, x_n) :- Q_2(x_1, x_2, \ldots, x_n).$
How do we know this Datalog query is safe?
We are assuming Q_1 and Q_2 are union-compatible. Is that okay? (QED for Union)

The remaining operators:

DIFFERENCE: if the query Q is $Q_1 - Q_2$ then the equivalent safe, non-recursive Datalog program is:

$Q(x_1, x_2, \ldots, x_n) :- Q_1(x_1, x_2, \ldots, x_n), \neg Q_2(x_1, x_2, \ldots, x_n).$

How do we know that Q, Q1, and Q2 have same attribute and thus the same number of attributes?

How do we know that this is safe? (QED for Diff.)

PROJECT:

NATURAL JOIN:

SELECT:

Prove that the theorem holds for the remaining operators:

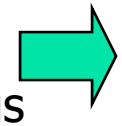
PROJECT:

NATURAL JOIN:

SELECT:

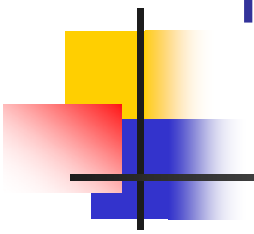
Reminder: Which Datalog language?

- Datalog – one rule, no negation, no recursion. (conjunctive queries)
- Datalog – multiple rules, no negation, no recursion.
- Datalog – multiple rules, no negation, with recursion. *Recursion but not relationally complete.*



- Datalog – multiple rules, with negation, no recursion. *Relationally complete but no recursion.*
- Datalog – multiple rules, with negation, with recursion. *Relationally complete with recursion but some queries are ambiguous!*

Prove safe, non-recursive Datalog is contained in allowable domain calculus



- How would you get started?

(Reminder) Datalog syntax

Atomic formulas:

$R(y_1, y_2, \ldots, y_k)$ – a predicate formula

$x = y$ (Note: this is syntatic sugar for equal(x,y).)

$R(v_1, v_2, \ldots, v_k)$ – a ground atomic formula

Literal:

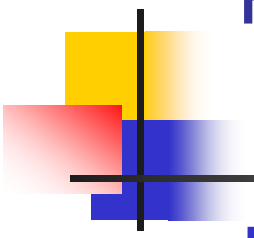
an atomic formula (positive literal) or the negation of an atomic formula (negative literal)

Clause (Datalog rule):

L :- $L_1, L_2, \ldots, L_n$.

where L is a predicate formula and Li, i = 1, …, n, is a positive or negative literal.

Prove safe, non-recursive Datalog is contained in allowable domain calculus



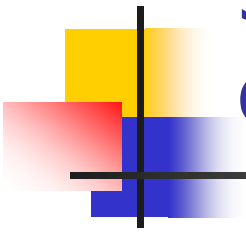
- Induction on the number of rules that appear in the Datalog program that have the relation symbol R in it's head.
- Base case: In the book, they use the case of zero rules. This means that the query answer is simply a relation from the existing database.
- Inductive step: Assume that we can express any Datalog program with q rules in an equivalent allowable domain calculus expression. Consider a Datalog program consisting of q+1 rules.
- Choose a rule and construct an allowable domain calculus expression.

Proving Safe Datalog is contained in Allowable Domain Calculus

Inductive step:

- introduce new variables – as many as there are attributes in the query answer. introduce $x_i=y_j$ (for repeated variables that occur in literal R) and $x_i=v$ (for constants that appear in literal R)
- Introduce "$\exists z_i$" for all variables in the body but not in head
- Introduce R1(…)^R2(…)^¬R3(…) to represent the body of the rule
- Find all other rules with same head, construct a body (as above), and then create one big disjunction.

3. Prove allowable domain relational calculus is contained in relational algebra



- This proof may be a little more interesting than the first two. In order to construct a relational algebra expression, you need to build some useful relations – to be used as input to relational algebra operators – based on what is present in the domain calculus expression.

Proving Allowable Domain Calculus is contained in Relational Algebra

- Induction on the number of logical connectors in the allowed domain calculus formula.
- Minimal set of logical connectors ($\neg$, v, $\exists$)
- For each $R(A_1, \ldots, A_m)$ construct an expression that gives the active domain of R.
- Do that for all R. Construct a dom. calculus Fdom(x) expression that comprises the union of all such domains or constants from the query.
- $\{ x_1, \ldots, x_n \mid F \text{ ^ } Fdom(x_1) \text{^ } \ldots \text{ ^ } Fdom(x_n)\}$
- We know how to construct rel. alg expressions for Fdom. We form the cross product of Reldom(F) n times and intersect is with the rel. alg. expression we need to express F – the original expression in Q.

Rest of the proof sketch

Base case: zero logical connectors. Then F is just one relation predicate. The rel alg expression is π ...(σ...R) to accommodate any constants or repeated variables in R(x1, …, xn) and to account for R having more variables than the desired query answer.

Induction: Assume it's true for q logical connectors.

$F_1 \vee F_2$: use π...($E_1 \cap RelDom(F_1)^{n-m}$) U π ...($E_2 \cap RelDom(F_2)^{n-k}$)

$\neg F$: use $RelDom(F)^n - E$

$\exists x\ (F)$: use π...(E)