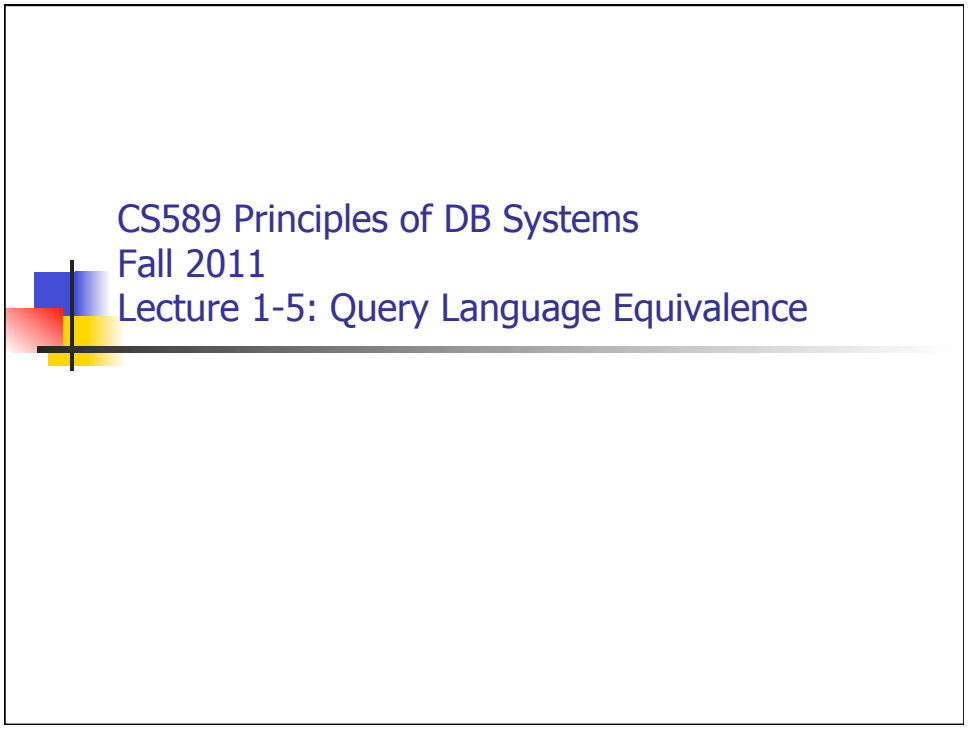
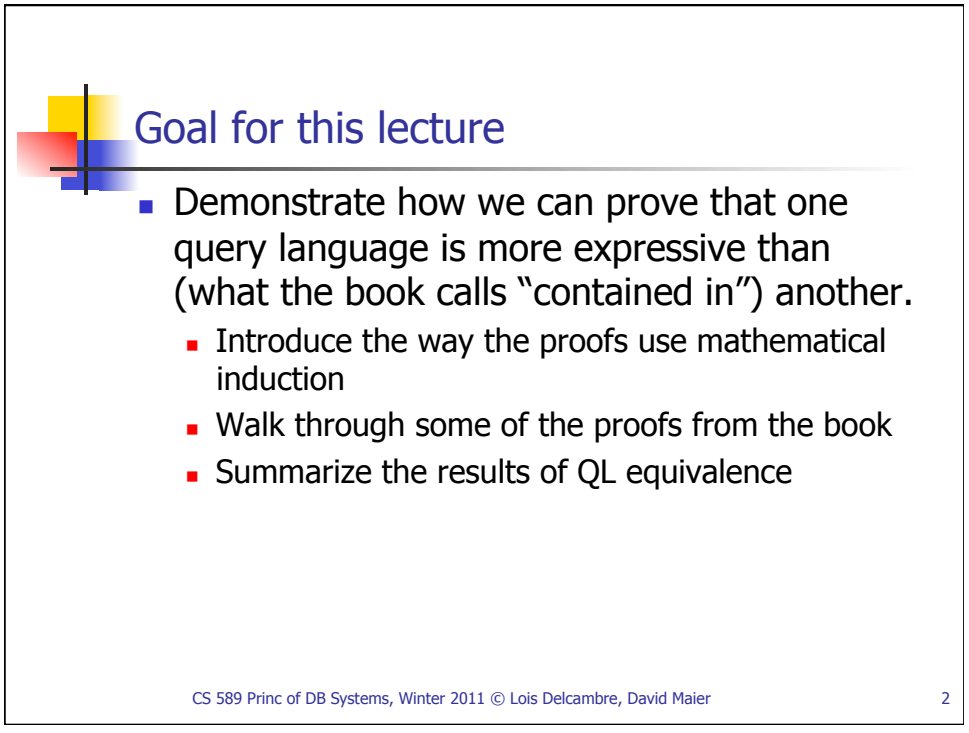

CS589 Principles of DB Systems
Fall 2011
Lecture 1-5: Query Language Equivalence


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
CS 589 Princ of DB Systems, Winter 2011 © Lois Delcambre, David Maier
2

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 with negation is contained in allowable domain calculus
3. Prove allowable domain 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)

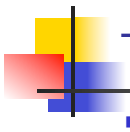
Minimal set of operators (with the full expressive power of the relational algebra):
$\cup$, $-$, π, $\bowtie$, σ
(By the way, how would you prove that this is, in fact, a minimal set of operators?)

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?

$A(x_1, x_2, \ldots, x_n)$:- $R(x_1, x_2, \ldots, x_n)$.



The induction step

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

  One of the operators in our minimal set.
  So proceed by cases

Union case

The query expression Q is $Q_1 \cup Q_2$

Q_1 and Q_2 must each have q or fewer operators

- Let P_1 be a safe program for Q_1 that gives answers via $A1(x_1, x_2, \ldots, x_n)$
- Let P_2 be a safe program for Q_2 that gives answers via $A1(x_1, x_2, \ldots, x_n)$

Create a Datalog program P for Q from P1, P2 and the two additional rules

$A(x_1, x_2, \ldots, x_n)$:- $A1(x_1, x_2, \ldots, x_n)$.
$A(x_1, x_2, \ldots, x_n)$:- $A2(x_1, x_2, \ldots, x_n)$.
How do we know P is safe?

Detail

We need to assume that head symbols in P_1 and P_2 are disjoint

For example, P_2 doesn't use A1

Difference case

If query Q is $Q_1 - Q_2$, with P_1 and P_2 as before, then the equivalent safe, non-recursive Datalog program is $P = P_1 + P_2 +$

$A(x_1, x_2, \ldots, x_n)$:- $A1(x_1, x_2, \ldots, x_n)$, $\neg A2(x_1, x_2, \ldots, x_n)$.

Is this safe?

Discuss the Remaining Cases

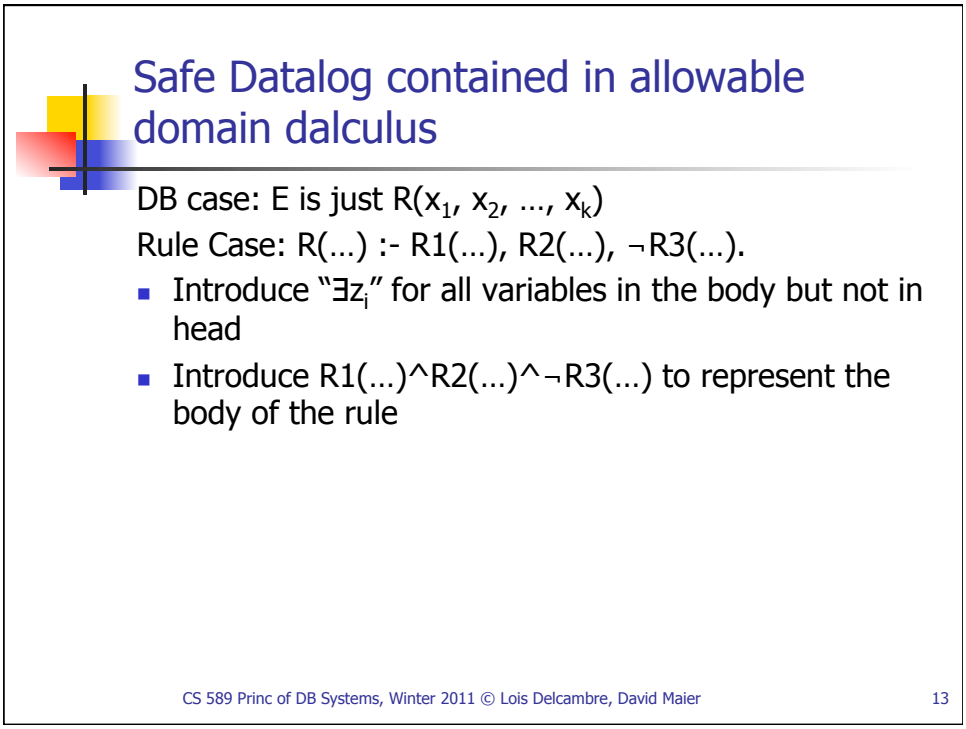
PROJECT:

NATURAL JOIN:

SELECT:

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

- Book starts from program P and a goal
  : - $R(y_1, y_2, \ldots, y_k)$.
- Assumes R is the only relation symbol in the head of rules.
- Structures proof as an induction – doesn't really need to be
- Need to deal with constants and repeated variables in the goal: :- R(y, 5, w, y).
  - Create expression F_R to handle these constraints
  - $\{x_1, x_2, x_3, x_4 \mid x_1 = x_4 \wedge x_2 = 5 \wedge F_Q\}$
- Construct an expression $E(x_1, x_2, \ldots, x_k)$ for facts in DB plus each rule and 'or' them together to get F_Q

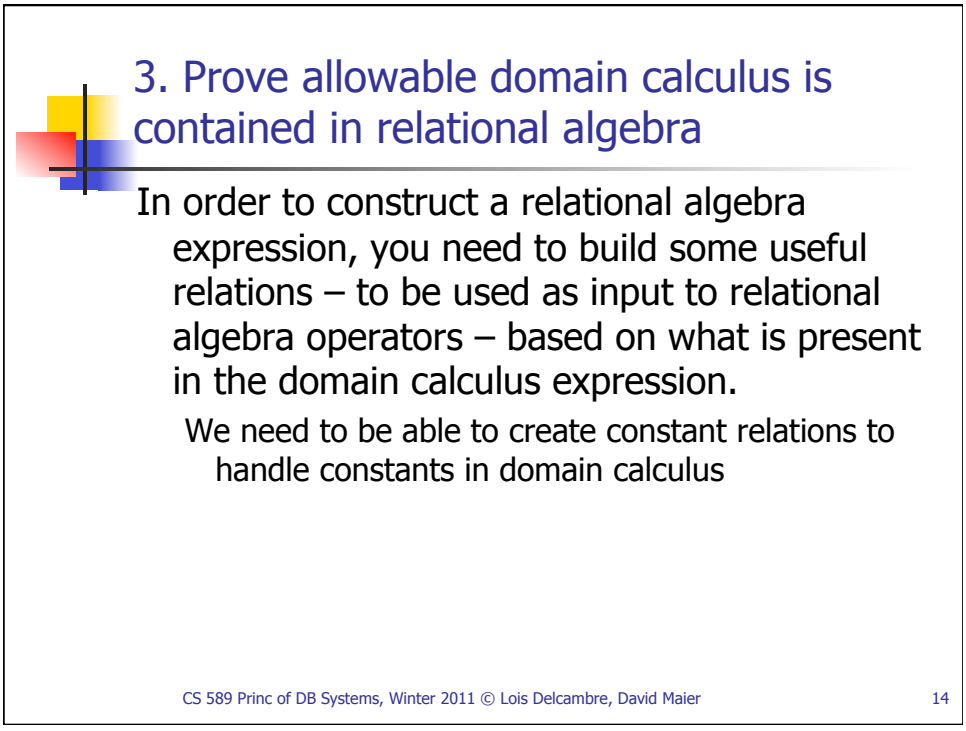


Safe Datalog contained in allowable domain dalculus

DB case: E is just $R(x_1, x_2, \ldots, x_k)$

Rule Case: R(…) :- R1(…), R2(…), ¬R3(…).

- 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



3. Prove allowable domain calculus is contained in relational algebra

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.

We need to be able to create constant relations to handle constants in domain calculus

Proving Allowable Domain Calculus is contained in Relational Algebra

- Induction on the number of logical connectors in the allowed domain calculus formula Q = $\{x_1, x_2, \ldots, x_n \mid F(x_1, x_2, \ldots, x_n)\}$
- F uses minimal set of logical connectors ($\neg$, $\vee$, $\exists$)
- We can construct a relational expression RelDom(F) that returns a one-attribute relation with all the values that a variable in Q can take on
  - Suppose F mentions R(A, B, C) and S(C, D) plus constants 17 and 23
  - Then RelDom(F) = $\pi_A(R) \cup \pi_B(R) \cup \pi_C(R) \cup \pi_C(S) \cup \pi_D(S) \cup \{<17>, <23>\}$
  - RelDom(F)^i is cross product of i copies of RelDom: relation with all possible i-tuples.

Rest of the proof sketch

Base case: zero logical connectors. Then F is just one relation predicate. The algebra expression is π ...(σ...R) to accommodate any constants or repeated variables in R(x_1, ..., x_n) and to account for R having more variables than the desired query answer.

Induction: Based on structure of F

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

$\neg F_1$: RelDom(F)^n – E

$\exists x\ (F_1)$: π...(E)