# CSE 544: Lecture 11
# Theory

Monday, May 3, 2004

## Query Minimization

**Definition** A conjunctive query q is minimal if for every other conjunctive query q' s.t. q ≡ q', q' has at least as many predicates ('subgoals') as q

Are these queries minimal ?

q(x) :- R(x,y), R(y,z), R(x,x)

q(x) :- R(x,y), R(y,z), R(x,'Alice')

## Query Minimization

- Query minimization algorithm

  Choose a subgoal g of q
  Remove g: let q' be the new query
  We already know q ⊆ q' (why ?)
  If q' ⊆ q then permanently remove g

- Notice: the order in which we inspect subgoals doesn't matter

## Query Minimization In Practice

- No database system today performs minimization !!!
- Reason:
  - It's hard (NP-complete)
  - Users don't write non-minimal queries
- However, non-minimal queries arise when using views intensively

## Query Minimization for Views

```
CREATE VIEW HappyBoaters

  SELECT DISTINCT E1.name, E1.manager
  FROM Employee E1, Employee E2
  WHERE E1.manager = E2.name
        and E1.boater='YES'
        and E2.boater='YES'
```

This query is minimal

## Query Minimization for Views

Now compute the Very-Happy-Boaters

```
SELECT DISTINCT  H1.name
FROM HappyBoaters H1, HappyBoaters H2
WHERE H1.manager = H2.name
```

This query is also minimal

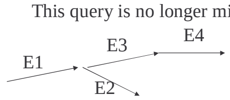
What happens in SQL when we run a query on a view ?

## Query Minimization for Views

View Expansion

```
SELECT DISTINCT  E1.name
FROM Employee E1, Employee E2, Employee E3, Empolyee E4
WHERE E1.manager = E2.name and E1.boater = 'YES' and E2.boater = 'YES'
    and E3.manager = E4.name and E3.boater = 'YES' and E4.boater = 'YES'
    and E1.manager = E3.name
```

This query is no longer minimal !




E2 is redundant

## Monotone Queries

**Definition** A query q is monotone if:
For every two databases D, D'
if $D \subseteq D'$ then $q(D) \subseteq q(D')$

Which queries below are monotone ?

$\varphi \equiv \exists x.R(x,x)$

$\varphi \equiv \exists x.\exists y.\exists z.\exists u.(R(x,y) \wedge R(y,z) \wedge R(z,u))$

$\varphi \equiv \exists x.\forall y.R(x,y)$

## Monotone Queries

- **Theorem**. Every conjunctive query is monotone
- Stronger: every UCQ query is monotone

## How To Impress Your Students Or Your Boss

- Find all drinkers that like some beer that is not served by the bar "Black Cat"

```
SELECT L.drinker
FROM Likes L
WHERE L.beer not in (SELECT S.beer
                     FROM Serves S
                     WHERE S.bar = 'Black Cat')
```

- Can you write as a simple SELECT-FROM-WHERE (I.e. without a subquery) ?

## Expressive Power of FO

- The following queries cannot be expressed in FO:
- Transitive closure:
  - $\forall x.\forall y.$ there exists $x_1, ..., x_n$ s.t.
    $R(x,x_1) \wedge R(x_1,x_2) \wedge ... \wedge R(x_{n-1},x_n) \wedge R(x_n,y)$
- Parity: the number of edges in R is even

## Datalog

- Adds recursion, so we can compute transitive closure
- A datalog program (query) consists of several datalog rules:

$P_1(t_1)$ :- $body_1$
$P_2(t_2)$ :- $body_2$
. . .
$P_n(t_n)$ :- $body_n$

## Datalog

Terminology:

- EDB = extensional database predicates
  - The database predicates
- IDB = intentional database predicates
  - The new predicates constructed by the program

## Datalog

Employee(x), ManagedBy(x,y), Manager(y)

All higher level managers that are employees:

EDBs

```
HMngr(x)  :- Manager(x), ManagedBy(y,x), ManagedBy(z,y)
Answer(x) :- HMngr(x), Employee(x)
```

IDBs

## Datalog

Employee(x), ManagedBy(x,y), Manager(y)

All persons:

```
Person(x) :- Manager(x)
Person(x) :- Employee(x)
```

Manger ∪ Employee

## Unfolding non-recursive rules

Graph: R(x,y)

```
P(x,y) :- R(x,u), R(u,v), R(v,y)
A(x,y) :- P(x,u), P(u,y)
```

Can "unfold" it into:

```
A(x,y) :- R(x,u), R(u,v), R(v,w), R(w,m), R(m,n), R(n,y)
```

## Unfolding non-recursive rules

Graph: R(x,y)

```
P(x,y) :- R(x,y)
P(x,y) :- R(x,u), R(u,y)
A(x,y) :- P(x,y)
```

Now the unfolding has a union:

A(x,y) :- R(x,y) $\vee$ $\exists$u(R(x,u) $\wedge$ R(u,y))

## Recursion in Datalog

Graph: R(x,y)

Transitive closure:

```
P(x,y) :- R(x,y)
P(x,y) :- P(x,u), R(u,y)
```

Transitive closure:

```
P(x,y) :- R(x,y)
P(x,y) :- P(x,u), P(u,y)
```

## Recursion in Datalog

Boolean trees:
Leaf0(x), Leaf1(x),
AND(x, $y_1$, $y_2$), OR(x, $y_1$, $y_2$),
Root(x)

- Write a program that computes:
  Answer() :- true if the root node is 1

## Recursion in Datalog

| | |
|---|---|
| One(x) | :- Leaf1(x) |
| One(x) | :- AND(x, $y_1$, $y_2$), One($y_1$), One($y_2$) |
| One(x) | :- OR(x, $y_1$, $y_2$), One($y_1$) |
| One(x) | :- OR(x, $y_1$, $y_2$), One($y_2$) |
| Answer() | :- Root(x), One(x) |

## Exercise

Boolean trees:
Leaf0(x), Leaf1(x),
AND(x, $y_1$, $y_2$), OR(x, $y_1$, $y_2$), Not(x,y),
Root(x)

- **Hint**: compute both One(x) and Zero(x) here you need to use Leaf0

## Variants of Datalog

| | without recursion | with recursion |
|---|---|---|
| without ¬ | Non-recursive Datalog = UCQ (why ?) | **Datalog** |
| with ¬ | Non-recursive Datalog¬ = FO | Datalog¬ |

## Non-recursive Datalog

- Union of Conjunctive Queries = UCQ
  - Containment is decidable, and NP-complete
- Non-recursive Datalog
  - Is equivalent to UCQ
  - Hence containment is decidable here too
  - Is it still NP-complete ?

## Non-recursive Datalog

- A non-recursive datalog:

$T_1$(x,y) :- R(x,u), R(u,y)
$T_2$(x,y) :- $T_1$(x,u), $T_1$(u,y)
. . .
$T_n$(x,y) :- $T_{n-1}$ (x,u), $T_{n-1}$ (u,y)
Answer(x,y) :- $T_n$(x,y)

- Its unfolding as a CQ:

Anser(x,y) :- R(x,$u_1$), R($u_1$, $u_2$), R($u_2$, $u_3$), . . . R($u_m$, y)

- How big is this query ?

## Query Complexity

- Given a query $\varphi$ in FO
- And given a model $\mathbf{D} = (D, R_1^D, \ldots, R_k^D)$
- What is the complexity of computing the answer $\varphi(D)$

## Query Complexity

Vardi's classification:

**Data Complexity:**
- Fix $\varphi$. Compute $\varphi(D)$ as a function of $|D|$

**Query Complexity:**
- Fix D. Compute $\varphi(D)$ as a function of $|\varphi|$

**Combined Complexity:**
- Compute $\varphi(D)$ as a function of $|D|$ and $|\varphi|$

Which is the most important in databases ?

## Example

$\varphi(x) \equiv \exists u.(R(u,x) \wedge \forall y.(\exists v.S(y,v) \Rightarrow \neg R(x,y)))$

R =

| | |
|---|---|
| 3 | 8 |
| 7 | 5 |
| 0 | 8 |
| 09 | 7 |
| 6 | 9 |
| 7 | 6 |
| 89 | 8 |
| 98 | 7 |
| 4 | 0 |

S =

| | |
|---|---|
| 43 | 4 |
| 5 | 58 |
| 8 | 6 |
| 9 | 79 |
| 6 | 67 |
| 4 | 7 |
| 6 | 8 |

How do we proceed ?

## General Evaluation Algorithm

**for** every subexpression $\varphi_i$ of $\varphi$, $(i = 1, \ldots, m)$
  compute the answer to $\varphi_i$ as a table $T_i(x_1, \ldots, x_n)$
**return** $T_m$

**Theorem**. If $\varphi$ has k variables then one can compute $\varphi(D)$ in time $O(|\varphi| * |D|^k)$

Data Complexity = $O(|D|^k)$ = in PTIME
Query Complexity = $O(|\varphi| * c^k)$ = in EXPTIME

## General Evaluation Algorithm

Example: $\varphi(x) \equiv \exists u.(R(u,x) \wedge \forall y.(\exists v.S(y,v) \Rightarrow \neg R(x,y)))$

$\varphi_1(u,x) \equiv R(u,x)$
$\varphi_2(y,v) \equiv S(y,v)$
$\varphi_3(x,y) \equiv \neg R(x,y)$
$\varphi_4(y) \equiv \exists v.\varphi_2(y,v)$
$\varphi_5(x,y) \equiv \varphi_4(y) \Rightarrow \varphi_3(x,y)$
$\varphi_6(x) \equiv \forall y.\ \varphi_5(x,y)$
$\varphi_7(u,x) \equiv \varphi_1(u,x) \wedge \varphi_6(x)$
$\varphi_8(x) \equiv \exists u.\ \varphi_7(u,x) \quad \equiv \varphi(x)$

## Complexity

**Theorem**. If $\varphi$ has k variables then one can compute $\varphi(D)$ in time $O(|\varphi| * |D|^k)$

**Remark**. The number of variables matters !

## Paying Attention to Variables

- Compute all chains of length m

$\text{Chain}_m(x,y)$ :- $R(x,u_1), R(u_1, u_2), R(u_2, u_3), \ldots R(u_{m-1}, y)$

- We used m+1 variables
- Can you rewrite it with fewer variables ?

## Counting Variables

- $FO^k$ = FO restricted to variables $x_1,\ldots, x_k$
- Write $\text{Chain}_m$ in $FO^3$:

$\text{Chain}_m(x,y)$ :- $\exists u.R(x,u)\wedge(\exists x.R(u, x)\wedge(\exists u.R(x,u)\ldots\wedge(\exists u.\ R(u, y)\ldots))$

## Query Complexity

- Note: it suffices to investigate boolean queries only
  - If non-boolean, do this:

**for** $a_1$ in D, …, $a_k$ in D<br>
    **if** $(a_1, \ldots, a_k)$ in $\varphi(D)$ /* this is a boolean query */<br>
        **then** output $(a_1, \ldots, a_k)$

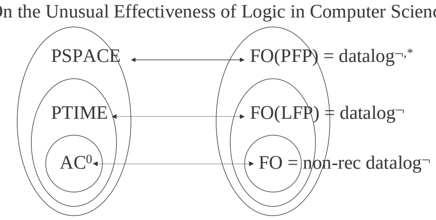
## Computational Complexity Classes

Recall computational complexity classes:

- $AC^0$
- LOGSPACE
- NLOGSPACE
- PTIME
- NP
- PSPACE
- EXPTIME
- EXPSPACE
- (Kalmar) Elementary Functions
- Turing Computable functions

## Data Complexity of Query Languages

Paper: On the Unusual Effectiveness of Logic in Computer Science




Important: the more complex a QL, the harder it is to optimize

## Views

Employee(x), ManagedBy(x,y), Manager(y)

Views:

- L(x,y) :- ManagedBy(x,u), ManagedBy(u,y)
- E(x,y) :- ManagedBy(x,y), Employee(y)

Query:

Q(x,y) :- ManagedBy(x,u), ManagedBy(u,v),<br>
ManagedBy(v,w), ManagedBy(w,y), Employee(y)

How can we answer Q if we only have L and E ?

## Views

- Query rewriting using views (when possible):

$$\boxed{Q(x,y) \text{ :- } L(x,u), L(u,y), E(v,y)}$$

- Query answering:
  - Sometimes we cannot express it in CQ or FO, but we can still answer it

## Views

Applications:

- Using advanced indexes
- Using replicated data
- Data integration [Ullman'99]