CSE 544
Theory of Query Languages

Tuesday, February 22nd, 2011

Outline

- Conjunctive queries; containment
- Datalog
- Query complexity

Conjunctive Queries

- A subset of Relational Calculus (=FO)
- Correspond to
  SELECT-DISTINCT-FROM-WHERE
- Most queries in practice are conjunctive
- Some optimizers handle only conjunctive queries
  - break larger queries into many CQs
- CQ’s have more positive theoretical properties than arbitrary queries

Conjunctive Queries

- **Definition** A conjunctive query is defined by:

$$\varphi ::= R(t_1, ..., t_k) \quad | \quad t_i = t_j \quad | \quad \varphi \wedge \varphi' \quad | \quad \exists x.\varphi$$

- missing are $\forall$, $\vee$, $\neg$

Conjunctive Queries, CQ

- Example of CQ

$$q(x,y) = \exists z.(R(x,z) \wedge \exists u.(R(z,u) \wedge R(u,y)))$$

$$q(x) = \exists z.\exists u.(R(x,z) \wedge R(z,u) \wedge R(u,y))$$

- Examples of non-CQ:

$$q(x,y) = \forall z.(R(x,z) \rightarrow R(y,z))$$

$$q(x) = T(x) \vee \exists z.S(x,z)$$

Conjunctive Queries

- Any CQ query can be written as:

$$q(x_1,\ldots,x_n) = \exists y_1.\exists y_2\ldots\exists y_p.(R_1(t_{11},\ldots,t_{1m}) \wedge \ldots \wedge R_k(t_{k1},\ldots,t_{km}))$$

(i.e. all quantifiers are at the beginning)

- Same in **Datalog** notation:

Datalog rule

$$\underbrace{q(x_1,\ldots,x_n)}_{\text{head}} \quad :- \quad \underbrace{R_1(t_{11},\ldots,t_{1m}), \ldots, R_k(t_{k1},\ldots,t_{km}))}_{\text{body}}$$

Examples

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

- Find all employees having the same manager as "Smith":

A(x) :- ManagedBy("Smith",y), ManagedBy(x,y)

Examples

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

- Find all employees having the same director as Smith:

A(x) :- ManagedBy("Smith",y), ManagedBy(y,z), ManagedBy(x,u), ManagedBy(u,z)

CQs are useful in practice

CQ and SQL

CQ:

```
A(x) :-  ManagedBy("Smith",y),   ManagedBy(x,y)
```

SQL:

Notice "distinct"

```
select distinct m2.name
from ManagedBy m1, ManagedBy m2
where m1.name="Smith" AND
        m1.manager=m2.manager
```

CQ and SQL

- Are CQ queries precisely the SELECT-DISTINCT-FROM-WHERE queries ?

CQ and SQL

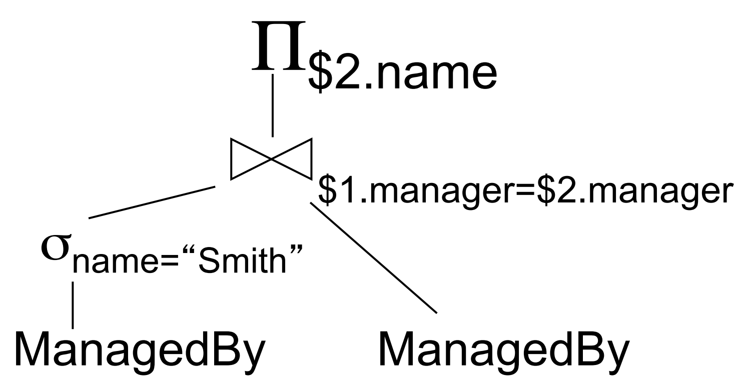
- Are CQ queries precisely the SELECT-DISTINCT-FROM-WHERE queries ?
- No: CQ queries do not allow $<$, $\leq$, $\neq$
- But we can extend CQ with inequality predicates, and usually write the extended language as $CQ^{<}$, etc

CQ and RA

Relational Algebra:

- CQ correspond precisely to σ_C, Π_A, $\times$ (missing: $\cup$, $-$) and where C has only =

A(x) :- ManagedBy("Smith",y), ManagedBy(x,y)



Extensions of CQ

$CQ^{\neq}$

Find managers that manage at least 2 employees

$$A(y) \text{ :- } ManagedBy(x,y), \quad ManagedBy(z,y), \; x \neq z$$

Extensions of CQ

$CQ^{<}$

Find employees earning more than their manager:

$$A(y) \text{ :- } ManagedBy(x,y), Salary(x,u), Salary(y,v), u>v$$

Extensions of CQ

CQ$^{\neg}$: negation applied only to one atom

Find people sharing the same office with Alice, but not the same manager:

```
A(y) :-  Office("Alice",u), Office(y,u),
           ManagedBy("Alice",x), ¬ManagedBy(x,y)
```

Extensions of CQ

UCQ Union of conjunctive queries

Datalog:

```
A(name) :-  Employee(name, dept, age, salary), age > 50
A(name) :-  RetiredEmployee(name, address)
```

Datalog notation is very convenient
for expressing unions (no need for $\vee$)

Summary of Extensions of CQ

- CQ
- $CQ^{\neq}$
- CQ<
- UCQ
- $CQ^{\neg}$

- Which of these classes contain only monotone queries ?

Query Equivalence and Containment

- Justified by optimization needs
- Intensively studied since 1977

Query Equivalence

- Queries q_1 and q_2 are **equivalent** if for every database **D**, $q_1(\mathbf{D}) = q_2(\mathbf{D})$.
- Notation: $q_1 \equiv q_2$

Query Containment

- Query q_1 is **contained** in q_2 if for every database **D**, $q_1(\mathbf{D}) \subseteq q_2(\mathbf{D})$.
- q_1 and q_2 are **equivalent** if for every database **D**, $q_1(\mathbf{D}) = q_2(\mathbf{D})$

- Notation: $q_1 \subseteq q_2$, $q_1 \equiv q_2$

- Obviously: $q_1 \subseteq q_2$ and $q_2 \subseteq q_1$ iff $q_1 \equiv q_2$
- Conversely: $q_1 \wedge q_2 \equiv q_1$ iff $q_1 \subseteq q_2$

We will study the containment problem only.

Examples of Query Containments

Is $q_1 \subseteq q_2$?

$q_1(x)$:- $R(x,u), R(u,v), R(v,w)$
$q_2(x)$:- $R(x,u), R(u,v)$

Examples of Query Containments

Is $q_1 \subseteq q_2$?

$q_1(x)$:- $R(x,u)$, $R(u,v)$, $R(v,x)$
$q_2(x)$:- $R(x,u)$, $R(u,x)$

Examples of Query Containments

Is $q_1 \subseteq q_2$?

$q_1(x)$:- $R(x,u)$, $R(u,u)$
$q_2(x)$:- $R(x,u)$, $R(u,v)$, $R(v,w)$

Examples of Query Containments

Is $q_1 \subseteq q_2$?

$q_1(x)$:- R(x,u), R(u,"Smith")
$q_2(x)$:- R(x,u), R(u,v)

Query Containment

- **Theorem** Query containment for FO is undecidable

- **Theorem** Query containment for CQ is decidable and NP-complete.

Trakhtenbrot's Theorem

Definition A sentence φ, is called *finitely satisfiable* if there exists a finite database instance D s.t. D |= φ

Satisfiable:

$\exists x.\exists y.\forall z.(R(x,z) \rightarrow R(y,z))$

$\exists x.\exists y.T(x) \vee \exists z.S(x,z)$

Unsatisfiable:

$\forall x.\forall y.\forall z.(R(x,y) \wedge R(x,z) \rightarrow y=z)$

$\wedge \exists y. \forall x. \text{ not } R(x,y)$

Theorem The following problem is undecidable:
Given FO sentence φ, check if φ is finitely satisfiable

Query Containment

- **Theorem** Query containment for FO is undecidable

- **Proof**: By reduction from the finite satisfiability problem:
- Given a sentence φ, define two queries: $q1(x) = R(x)$, and $q2(x) = R(x) \wedge \varphi$
- Then $q1 \subseteq q2$ iff φ is not finitely satisfiable

Query Containment Algorithm

How to check $q_1 \subseteq q_2$

- **Canonical database** for q_1 is:

  $\mathbf{D_{q1}} = (D, R_1^D, \ldots, R_k^D)$
  - D = all variables and constants in q_1
  - $R_1^D, \ldots, R_k^D$ = the body of q_1
- **Canonical tuple** for q_1 is:

  t_{q1} (the head of q_1)

Examples of Canonical Databases

$\boxed{q1(x,y) \text{ :- } R(x,u),R(v,u),R(v,y)}$

- Canonical database: $\mathbf{D}_{q1} = (D, R^D)$
  - $D=\{x,y,u,v\}$
  - $R^D =$

x	u
v	u
v	y

- Canonical tuple: $t_{q1} = (x,y)$

Examples of Canonical Databases

q1(x) :- R(x,u), R(u,"Smith"), R(u,"Fred"), R(u, u)

- $\mathbf{D}_{q1}$ = (D, R)
  - D={x,u,"Smith","Fred"}
  - R =

x	u
u	"Smith"
u	"Fred"
u	u

- t_{q1} = (x)

Checking Containment

Theorem: $q_1 \subseteq q_2$ iff $t_{q1} \in q_2(\mathbf{D}_{q1})$.

Example:

$q_1(x,y)$:- $R(x,u),R(v,u),R(v,y)$
$q_2(x,y)$:- $R(x,u),R(v,u),R(v,w),R(t,w),R(t,y)$

- D={x,y,u,v}
- R = $\quad t_{q1} = (x,y)$

x	u
v	u
v	y

- Yes, $q_1 \subseteq q_2$

Query Homomorphisms

- A ***homomorphism*** $f : q_2 \rightarrow q_1$ is a function $f: var(q_2) \rightarrow var(q_1) \cup const(q_1)$ such that:
  - $f(body(q_2)) \subseteq body(q_1)$
  - $f(t_{q1}) = t_{q2}$

The Homomorphism Theorem $q_1 \subseteq q_2$ iff there exists a homomorphism $f : q_2 \rightarrow q_1$

Example of Query Homeomorphism

$\mathrm{var}(q_1) = \{x, u, v, y\}$

$\mathrm{var}(q_2) = \{x, u, v, w, t, y\}$

$q_1(x,y) \text{ :- } R(x,u),R(v,u),R(v,y)$

$q_2(x,y) \text{ :- } R(x,u),R(v,u),R(v,w),R(t,w),R(t,y)$

Therefore $q_1 \subseteq q_2$

Example of Query Homeomorphism

$\text{var}(q_1) \cup \text{const}(q_1) = \{x, u, \text{"Smith"}\}$

$\text{var}(q_2) = \{x, u, v, w\}$

$q_1(x)$:- $R(x,u)$, $R(u,\text{"Smith"})$, $R(u,\text{"Fred"})$, $R(u, u)$

$q_2(x)$:- $R(x,u)$, $R(u,v)$, $R(u,\text{"Smith"})$, $R(w,u)$

Therefore $q_1 \subseteq q_2$

The Complexity

Theorem Checking containment of two CQ queries is NP-complete

Containment for extensions of CQ

- CQ -- NP complete
- $CQ^{\neq}$ -- ??
- $CQ^{<}$ -- ??
- UCQ -- ??
- $CQ^{\neg}$ -- ??
- Relational Calculus -- undecidable

Query Containment for UCQ

$q_1 \cup q_2 \cup q_3 \cup \ldots . \subseteq q_1' \cup q_2' \cup q_3' \cup \ldots .$

Notice: $q_1 \cup q_2 \cup q_3 \cup \ldots . \subseteq q$ iff
$q_1 \subseteq q$ and $q_2 \subseteq q$ and $q_3 \subseteq q$ and ….

Theorem $q \subseteq q_1' \cup q_2' \cup q_3' \cup \ldots .$ Iff there exists some k such that $q \subseteq q_k'$

It follows that containment for UCQ is decidable, NP-complete.

Query Containment for $CQ^<$

$q_1()$:- R(x,y), R(y,x)
$q_2()$:- R(x,y), x <= y

$q_1 \subseteq q_2$ although there is no homomorphism !

To check containment do this:

- Consider all possible orderings of variables in q1
- For each of them check containment of q1 in q2
- If all hold, then $q_1 \subseteq q_2$

Still decidable, but harder than NP: now in Π^p_2

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

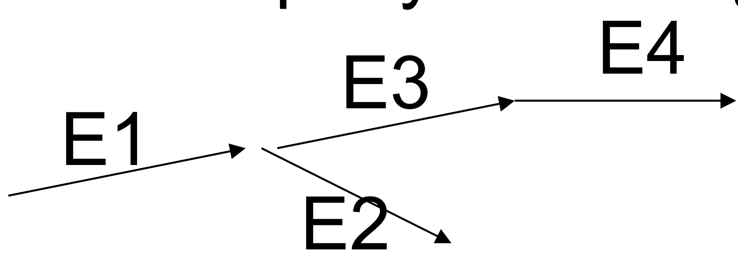
What happens in SQL when we run a query on a view ?

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

Expressive Power of FO

- The following queries cannot be expressed in Relational Calculus (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))

Non-recursive datalog = UCQ (why ?)

Recursion in Datalog

Example 1: transitive closure

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

Example 2: same generation

Graph: R(x,y)

```
Sg(x,x) :- R(x,y)
Sg(y,y) :- R(x,y)
Sg(x,y) :- R(x,u), Sg(u,v),R(v,y)
```

Recursion in Datalog

Example 3: value of a Boolean circuit with And/Not gates

Graph: And(y,z,x), Not(y,x), Value0(x), Value1(x)

```
isZero(x) :- Value0(x)
isOne(x)  :- Value1(x)
isZero(x) :- And(y,z,x),isZero(y)
isZero(x) :- And(y,z,x),isZero(z)
isOne(x)  :- And(y,z,x),isOne(y),isOne(z)
isZero(x) :- Not(y,x),isOne(y)
isOne(x)  :- Not(y,x),isZero(x)
```

Semantics of a Datalog Program

- Let:
  - R = the EDB predicates = some input instance D
  - S = the IDB predicates
- A datalog program P maps IDB predicate instances S to new IDB predicate instances S': S' = $P_D(S)$
- The function P_D is monotone (why ?)
- By Knaster-Tarski's fixpoint Theorem, P_D has a least fixpoint
- Definition: the meaning of P_D <u>*is*</u> the least fixpoint
- Alternatively: compute the meaning of P_D as follows:
  - S_0 = emptyset; $S_{k+1} = P(S_k)$
  - The meaning is: $P_D = S_0 \cup S_1 \cup S_2 \cup \ldots S_n \cup \ldots$

Evaluation of Datalog Programs

- Naïve evaluation algorithm:
  - Start with S = emptyset
  - Evaluate P on the EDB, and on the current IDB S; new-S=P(S); note: S $\subseteq$ new-S (why ?)
  - Replace S with new-S; repeat;
  - Stop when S = new-S
  - What is the complexity ?
- Semi-naïve evaluation algorithm:
  - Keep track of ΔS = new-S - S
  - Improve somewhat the computation of P(S)

What is the complexity of the naïve/ semi-naïve algorithm ?

Extensions of datalog with negation

- Problems with negation:

S(x) :- R(x), not T(x)
T(x) :- R(x), not S(x)

- There is no minimal fixpoint

Extensions of datalog with negation

- **Solution 1: Stratified Datalog¬**
  - Insist that the program be *stratified*: rules are partitioned into strata, and an IDB predicate that occurs only in strata ≤ k may be negated in strata ≥ k+1
- **Solution 2: Inflationary-fixpoint Datalog¬**
  - Run the naïve algorithm substituting: S = S ∪ new-S
  - Always terminates (why ?)
- **Solution 3: Partial-fixpoint Datalog¬,***
  - Run the naïve algorithm, substituting S = new-S
  - May not terminate (but we can detect that – how ?)

Datalog¬

Example: complement transitive closure

Graph: R(x,y)

```
Tc(x,y) :- R(x,y)
Tc(x,y) :- Tc(x,u), R(u,y)
CTc(x,y) :- R(x,u),R(v,y),not Tc(x,y)
```

This is stratified datalog¬

Challenge: express CTc in inflationary datalog¬

Expressive Power

Theorem:
stratified-datalog$^{\neg}$ $\subsetneq$
inflationary-datalog$^{\neg}$ $\subsetneq$
partial-datalog$^{\neg}$

Variants of Datalog

	without recursion	with recursion
without ¬	Non-recursive Datalog = UCQ	**Datalog**
with ¬	Non-recursive Datalog$^{\neg}$ = FO	Datalog$^{\neg}$ (three variants)

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:

$$
\begin{aligned}
&T_1(x,y) \quad \text{:-} \;\; R(x,u), R(u,y)\\
&T_2(x,y) \quad \text{:-} \;\; T_1(x,u), T_1(u,y)\\
&\quad . \quad . \quad .\\
&T_n(x,y) \quad \text{:-} \;\; T_{n-1}(x,u), T_{n-1}(u,y)\\
&\text{Answer}(x,y) \text{ :- } T_n(x,y)
\end{aligned}
$$

- Its unfolding as a CQ:

$$\boxed{\text{Anser}(x,y) \quad \text{:-} \;\; R(x,u_1), R(u_1, u_2), R(u_2, u_3), \ldots R(u_m, y)}$$

- How big is this query ?

Non-recursive Datalog

- A non-recursive datalog:

$$
\begin{aligned}
&T_1(x,y) \quad \text{:-} \quad R(x,u), R(u,y) \\
&T_2(x,y) \quad \text{:-} \quad T_1(x,u), T_1(u,y) \\
&\quad . \quad . \quad . \\
&T_n(x,y) \quad \text{:-} \quad T_{n-1}(x,u), T_{n-1}(u,y) \\
&\text{Answer}(x,y) \text{ :- } T_n(x,y)
\end{aligned}
$$

- Its unfolding as a CQ:

$$\text{Anser}(x,y) \quad \text{:-} \quad R(x,u_1), R(u_1, u_2), R(u_2, u_3), \ldots R(u_m, y)$$

- How big is this query ?

Although non-recursive-datalog = UCQ,
the former is exponentially more concise

Query Complexity

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

Query Complexity

Vardi's taxonomy:

Data Complexity:

- Fix φ. 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 φ_i of φ, (i = 1, …, m)
  compute the answer to φ_i as a table $T_i(x_1, \ldots, x_n)$
return T_m

Theorem. If φ 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:

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

$$\begin{aligned}
\varphi_1(u,x) &\equiv R(u,x) \\
\varphi_2(y,v) &\equiv S(y,v) \\
\varphi_3(x,y) &\equiv \neg R(x,y) \\
\varphi_4(y) &\equiv \exists v.\varphi_2(y,v) \\
\varphi_5(x,y) &\equiv \varphi_4(y) \Rightarrow \varphi_3(x,y) \\
\varphi_6(x) &\equiv \forall y.\ \varphi_5(x,y) \\
\varphi_7(u,x) &\equiv \varphi_1(u,x) \wedge \varphi_6(x) \\
\varphi_8(x) &\equiv \exists u.\ \varphi_7(u,x) \qquad \equiv \varphi(x)
\end{aligned}$$

Complexity

Theorem. If φ 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

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

- 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 $Chain_m$ in FO^3:

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

Query Complexity

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

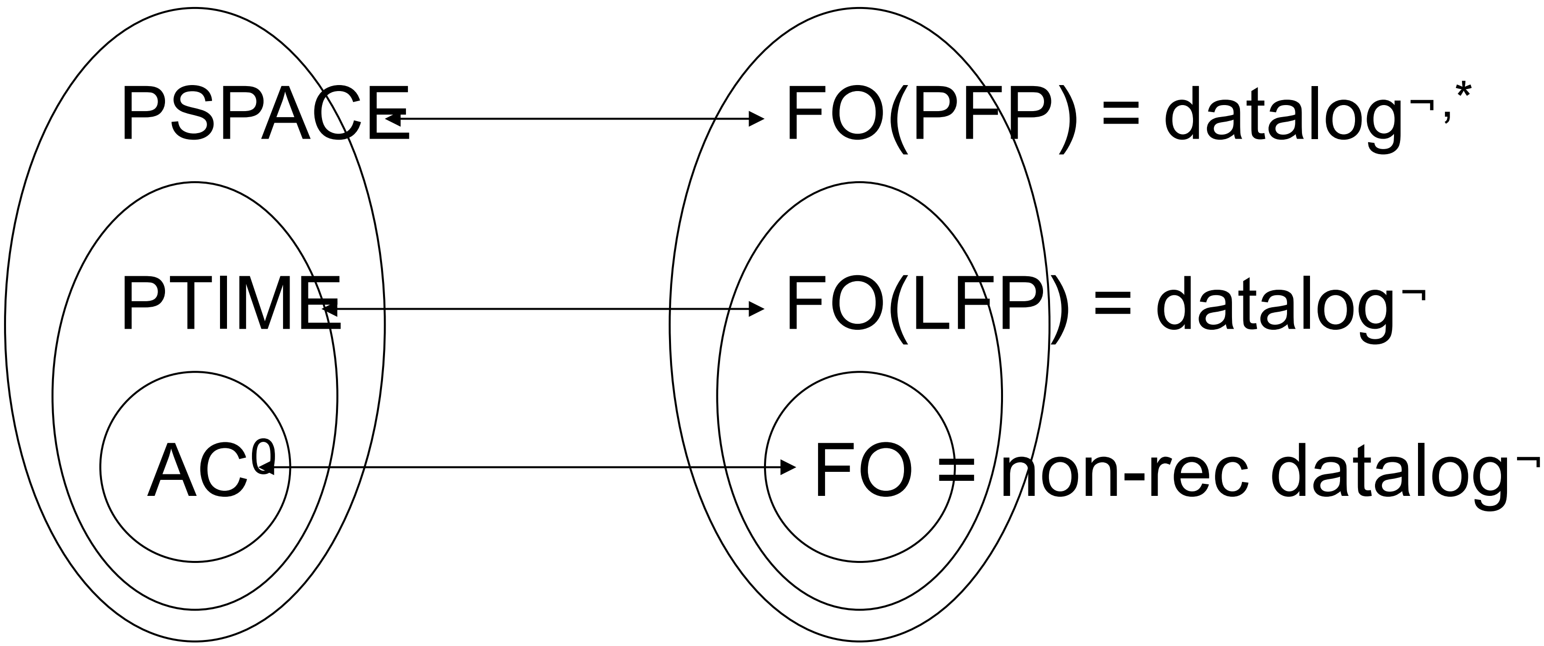
for a_1 in D, …, a_k in D
    if (a_1, …, a_k) in φ(D) /* this is a boolean query */
        then output (a_1, …, a_k)

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



The more complex a QL, the harder it is to optimize