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:

\[
\varphi ::= R(t_1, \ldots, t_k) \mid t_i = t_j \mid \varphi \land \varphi' \mid \exists x. \varphi
\]

• missing are $\forall$, $\lor$, $\neg$
Conjunctive Queries, CQ

• Example of CQ

\[ q(x,y) = \exists z.(R(x,z) \land \exists u.(R(z,u) \land R(u,y))) \]
\[ q(x) = \exists z.\exists u.(R(x,z) \land R(z,u) \land R(u,y)) \]

• Examples of non-CQ:

\[ q(x,y) = \forall z.(R(x,z) \rightarrow R(y,z)) \]
\[ q(x) = T(x) \lor \exists z.S(x,z) \]
Conjunctive Queries

• Any CQ query can be written as:

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

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

• Same in Datalog notation:

\[
q(x_1,\ldots,x_n) :- R_1(t_{11},\ldots,t_{1m}), \ldots , R_k(t_{k1},\ldots,t_{km}))
\]
Examples

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

• Find all employees having the same manager as “Smith”:

\[
A(x) :- \text{ManagedBy(“Smith”,y), ManagedBy(x,y)}
\]
Examples

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

• Find all employees having the same director as Smith:

\[
A(x) :- \text{ManagedBy("Smith",y), ManagedBy(y,z), ManagedBy(x,u), ManagedBy(u,z)}
\]

CQs are useful in practice
CQ and SQL

CQ:

\[ A(x) : \text{ManagedBy("Smith",y), ManagedBy(x,y)} \]

SQL:

```
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 <, ≤, ≠

• 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 $\sigma_C$, $\Pi_A$, $\times$
  (missing: $\cup$, $-$) and where $C$ has only $= $

A(x) :- ManagedBy(“Smith”,y), ManagedBy(x,y)
Extensions of CQ

\[ A(y) :- \text{ManagedBy}(x,y), \text{ManagedBy}(z,y), x \neq z \]

Find managers that manage at least 2 employees
Extensions of CQ

\[ \text{CQ}^< \]

Find employees earning more than their manager:

\[ A(y) : \text{ManagedBy}(x,y), \text{Salary}(x,u), \text{Salary}(y,v), u>v \]
Extensions of CQ

CQ⁻: negation applied only to one atom

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

\[
A(y) :- \text{Office(“Alice”,u), Office(y,u), ManagedBy(“Alice”,x), } \neg \text{ManagedBy(x,y)}
\]
Extensions of CQ

UCQ Union of conjunctive queries

Datalog:

\[
A(\text{name}) :- \text{Employee}(\text{name, dept, age, salary}), \text{age} > 50 \\
A(\text{name}) :- \text{RetiredEmployee}(\text{name, address})
\]

Datalog notation is very convenient for expressing unions (no need for \( \lor \) )
Summary of Extensions of CQ

- CQ
- CQ≠
- CQ<
- UCQ
- CQ⁻

- 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 $D$, $q_1(D) = q_2(D)$.

• Notation: $q_1 \equiv q_2$
Query Containment

• Query $q_1$ is **contained** in $q_2$ if for every database $D$, $q_1(D) \subseteq q_2(D)$.

• $q_1$ and $q_2$ are **equivalent** if for every database $D$, $q_1(D) = q_2(D)$

• 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: $q_1 \land q_2 \equiv q_1$ 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 \( \phi \), is called **finitely satisfiable** if there exists a finite database instance \( D \) s.t. \( D \models \phi \)

**Satisfiable:**
- \( \exists x. \exists y. \forall z. (R(x,z) \rightarrow R(y,z)) \)
- \( \exists x. \exists y. T(x) \lor \exists z. S(x,z) \)

**Unsatisfiable:**
- \( \forall x. \forall y. \forall z. (R(x,y) \land R(x,z) \rightarrow y = z) \)
- \( \land \exists y. \forall x. \neg R(x,y) \)

**Theorem** The following problem is undecidable:
Given FO sentence \( \phi \), check if \( \phi \) is finitely satisfiable
Query Containment

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

• **Proof**: By reduction from the finite satisfiability problem:
  • Given a sentence $\varphi$, define two queries:
    $q_1(x) = R(x)$, and $q_2(x) = R(x) \land \varphi$
  • Then $q_1 \subseteq q_2$ iff $\varphi$ is not finitely satisfiable
Query Containment Algorithm

How to check $q_1 \subseteq q_2$

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

- **Canonical tuple** for $q_1$ is:
  $t_{q_1}$ (the head of $q_1$)
Examples of Canonical Databases

q1(x,y) :- R(x,u),R(v,u),R(v,y)

- Canonical database: $D_{q1} = (D, R^D)$
  - $D=\{x,y,u,v\}$
  - $R^D = \begin{array}{|c|c|}
        x & u \\
        v & u \\
        v & y \\
        \end{array}$

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

\[ q_1(x) :- R(x,u), R(u,"Smith"), R(u,"Fred"), R(u, u) \]

- \( D_{q_1} = (D, R) \)
  - \( D = \{x, u, "Smith", "Fred"\} \)
  - \( R = \)
    
    | x   |   u      |
    |-----|----------|
    | u   | "Smith" |
    | u   | "Fred"  |
    | u   |   u      |

- \( t_{q_1} = (x) \)
Checking Containment

**Theorem:** $q_1 \subseteq q_2$ iff $t_{q_1} \in q_2(D_{q_1})$.

Example:

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

- $D=\{x,y,u,v\}$
- $R = t_{q_1} = (x,y)$

\[
\begin{array}{c|c}
  x & u \\
  v & u \\
  v & y \\
\end{array}
\]

- Yes, $q_1 \subseteq q_2$
Query Homomorphisms

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

**The Homomorphism Theorem** \( q_1 \subseteq q_2 \) iff there exists a homomorphism \( f : q_2 \rightarrow q_1 \)
Example of Query Homeomorphism

\[ \text{var}(q_1) = \{x, u, v, y\} \]

\[ \text{var}(q_2) = \{x, u, v, w, t, y\} \]

\[ q_1(x,y) :- R(x,u), R(v,u), R(v,y) \]

\[ q_2(x,y) :- R(x,u), R(v,u), R(v,w), R(t,w), R(t,y) \]

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 ≠ -- ??
- CQ^< -- ??
- UCQ -- ??
- CQ^- -- ??
- 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 \ldots

**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<

\[ q_1() \text{ :- } R(x,y), R(y,x) \]
\[ q_2() \text{ :- } R(x,y), x \leq y \]

\( q_1 \subseteq q_2 \) although there is no homomorphism!

To check containment do this:
- Consider all possible orderings of variables in \( q_1 \)
- For each of them check containment of \( q_1 \) in \( q_2 \)
- If all hold, then \( q_1 \subseteq q_2 \)

Still decidable, but harder than NP: now in \( \Pi^p_2 \)
Query Minimization

**Definition** A conjunctive query $q$ is minimal if for every other conjunctive query $q'$ s.t. $q \equiv q'$, $q'$ 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, Employee 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!

E1
  ↓
  E2
  ↓
  E3
  ↓
  E4

E2 is redundant
Expressive Power of FO

• The following queries cannot be expressed in Relational Calculus (FO):

• Transitive closure:
  – ∀x.∀y. there exists x₁, ..., xₙ s.t.
    R(x,x₁) ∧ R(x₁,x₂) ∧ ... ∧ R(xₙ₋₁,xₙ) ∧ R(xₙ,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:

\[ P_1(t_1) :- \text{body}_1 \]
\[ P_2(t_2) :- \text{body}_2 \]
\[ \ldots \]
\[ P_n(t_n) :- \text{body}_n \]
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:

HMngr(x) :- Manager(x), ManagedBy(y,x), ManagedBy(z,y)

Answer(x) :- HMngr(x), Employee(x)

EDBs

IDBs
Datalog

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

All persons:

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

Manager $\cup$ 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:  \( R(x,y) \)

\[
\begin{align*}
P(x,y) & : \ R(x,y) \\
P(x,y) & : \ R(x,u), \ R(u,y) \\
A(x,y) & : \ P(x,y)
\end{align*}
\]

Now the unfolding has a union:

\[
A(x,y) : \ R(x,y) \lor \exists u (R(x,u) \land R(u,y))
\]

Non-recursive datalog = UCQ  (why ?)
Recursion in Datalog

Example 1: transitive closure

Graph: \( R(x, y) \)

Transitive closure:

\[
\begin{align*}
P(x, y) & : \text{ R}(x, y) \\
P(x, y) & : P(x, u), R(u, y)
\end{align*}
\]

Transitive closure:

\[
\begin{align*}
P(x, y) & : \text{ R}(x, y) \\
P(x, y) & : P(x, u), P(u, y)
\end{align*}
\]
Recursion in Datalog

Example 2: same generation

Graph: \( R(x,y) \)

\[
\begin{align*}
Sg(x,x) & \ :- \ R(x,y) \\
Sg(y,y) & \ :- \ R(x,y) \\
Sg(x,y) & \ :- \ R(x,u), \ Sg(u,v), R(v,y)
\end{align*}
\]
Recursion in Datalog

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

Graph: \(\text{And}(y,z,x), \text{Not}(y,x), \text{Value0}(x), \text{Value1}(x)\)

\[
\begin{align*}
\text{isZero}(x) & : - \text{Value0}(x) \\
\text{isOne}(x) & : - \text{Value1}(x) \\
\text{isZero}(x) & : - \text{And}(y,z,x), \text{isZero}(y) \\
\text{isZero}(x) & : - \text{And}(y,z,x), \text{isZero}(z) \\
\text{isOne}(x) & : - \text{And}(y,z,x), \text{isOne}(y), \text{isOne}(z) \\
\text{isZero}(x) & : - \text{Not}(y,x), \text{isOne}(y) \\
\text{isOne}(x) & : - \text{Not}(y,x), \text{isZero}(x)
\end{align*}
\]
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 $P_D$ is the least fixpoint

• Alternatively: compute the meaning of $P_D$ as follows:
  – $S_0 =$ emptyset; $S_{k+1} = P(S_k)$
  – The meaning is: $P_D = S_0 \cup S_1 \cup S_2 \cup ... \cup S_n \cup ...$
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 $S = \text{new-S}$
  – What is the complexity ?

• Semi-naïve evaluation algorithm:
  – Keep track of $\Delta S = \text{new-S} - S$
  – 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:

  \[ S(x) :- R(x), \text{ not } T(x) \]
  \[ T(x) :- R(x), \text{ not } S(x) \]

• 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: \( S = S \cup \text{new-S} \)
  – Always terminates (why ?)

• **Solution 3: Partial-fixpoint Datalog**
  – Run the naïve algorithm, substituting \( S = \text{new-S} \)
  – May not terminate (but we can detect that – how ?)
Datalog^\neg

**Example:** complement transitive closure

Graph: $R(x,y)$

\[
\begin{align*}
Tc(x,y) & :\ - R(x,y) \\
Tc(x,y) & :\ - Tc(x,u), R(u,y) \\
CTc(x,y) & :\ - R(x,u), R(v,y), \text{not } Tc(x,y)
\end{align*}
\]

This is stratified datalog^\neg

Challenge: express CTc in inflationary datalog^\neg
Expressive Power

Theorem:
stratified-datalog $\not\subseteq$ inflationary-datalog $\not\subseteq$ partial-datalog
## Variants of Datalog

<table>
<thead>
<tr>
<th></th>
<th>without recursion</th>
<th>with recursion</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>without ( \neg )</strong></td>
<td>Non-recursive Datalog ( = \text{UCQ} )</td>
<td>Datalog</td>
</tr>
<tr>
<td><strong>with ( \neg )</strong></td>
<td>Non-recursive Datalog ( = \text{FO} )</td>
<td>Datalog( \neg ) (three variants)</td>
</tr>
</tbody>
</table>
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:
  
  \[
  T_1(x,y) \ :- \ R(x,u), R(u,y) \\
  T_2(x,y) \ :- \ T_1(x,u), T_1(u,y) \\
  \ldots \ldots \\
  T_n(x,y) \ :- \ T_{n-1}(x,u), T_{n-1}(u,y) \\
  Answer(x,y) \ :- \ T_n(x,y)
  \]

- Its unfolding as a CQ:

  \[
  Anser(x,y) \ :- \ R(x,u_1), R(u_1, u_2), R(u_2, u_3), \ldots R(u_m, y)
  \]

- How big is this query?
Non-recursive Datalog

• A non-recursive datalog:

\[
T_1(x, y) \ :- \ R(x, u), R(u, y) \\
T_2(x, y) \ :- \ T_1(x, u), T_1(u, y) \\
\ldots \\
T_n(x, y) \ :- \ T_{n-1}(x, u), T_{n-1}(u, y) \\
Answer(x, y) \ :- \ T_n(x, y)
\]

• Its unfolding as a CQ:

\[
Answer(x, y) \ :- \ R(x, u_1), R(u_1, u_2), R(u_2, u_3), \ldots R(u_m, y)
\]

• How big is this query?

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

• Given a query $\varphi$ 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 \( \varphi \). 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

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

R =

<table>
<thead>
<tr>
<th>3</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>5</td>
</tr>
<tr>
<td>0</td>
<td>8</td>
</tr>
<tr>
<td>09</td>
<td>7</td>
</tr>
<tr>
<td>6</td>
<td>9</td>
</tr>
<tr>
<td>7</td>
<td>6</td>
</tr>
<tr>
<td>89</td>
<td>8</td>
</tr>
<tr>
<td>98</td>
<td>7</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
</tr>
</tbody>
</table>

S =

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

How do we proceed?
General Evaluation Algorithm

\begin{quote}
\textbf{for} every subexpression $\varphi_i$ of $\varphi$, ($i = 1, \ldots, m$) \\
\hspace{1cm} compute the answer to $\varphi_i$ as a table $T_i(x_1, \ldots, x_n)$ \\
\textbf{return} $T_m$
\end{quote}

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

Data Complexity $= O(|D|^k) = \text{in PTIME}$
Query Complexity $= O(|\varphi|^*c^k) = \text{in EXPTIME}$
General Evaluation Algorithm

Example:

\[ \varphi(x) \equiv \exists u. (R(u,x) \land \forall y. (\exists v. S(y,v) \Rightarrow \neg R(x,y))) \]

\begin{align*}
\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) \land \varphi_6(x) \\
\varphi_8(x) & \equiv \exists u. \varphi_7(u,x) \equiv \varphi(x)
\end{align*}
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

- $F^{k}$ = FO restricted to variables $x_{1}, \ldots, x_{k}$

- Write $\text{Chain}_{m}$ in $\text{FO}^{3}$:

$$\text{Chain}_{m}(x,y) \iff \exists u. R(x,u) \land (\exists x. R(u, x) \land (\exists u. R(x,u) \ldots \land (\exists u. R(u, y) \ldots}))$$
Query Complexity

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

```python
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:

• $\text{AC}^0$
• $\text{LOGSPACE}$
• $\text{NLOGSPACE}$
• $\text{PTIME}$
• $\text{NP}$
• $\text{PSPACE}$
• $\text{EXPTIME}$
• $\text{EXPSPACE}$
• (Kalmar) Elementary Functions
• Turing Computable functions
Data Complexity of Query Languages

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