Database Design and Implementation

CS 645

Database theory
Theory problems in databases

- Expressiveness of languages
- Complexity of languages
- Static analysis of queries (for optimization)
- Views
Theory problems in databases

- Expressiveness of languages
  - Any query in L1 can be expressed in L2

- Complexity of languages
  - Bounds on resources required to evaluate any query in language L

- Static analysis of queries (for optimization)
  - Given q in L: is it minimal?
  - Given q1 and q2 in L: are they equivalent?

- Views
Crash review of complexity classes

- AC<sup>0</sup>
- L (LOGSPACE)
- NL (NLOGSPACE)
- NC
- NC
- P (PTIME)
- NP
- PSPACE
Crash review of complexity classes

**AC^0**
- Circuits of O(1) depth and polynomial size

**L (LOGSPACE)**
- Solvable in logarithmic (small) space

**NL (NLOGSPACE)**
- "YES" answers checkable in logarithmic space

**NC**
- Solvable efficiently (in polylogarithmic time) on parallel computers

**P (PTIME)**
- Solvable in polynomial time

**NP**
- "YES" answers checkable in polynomial time

**PSPACE**
- Solvable with polynomial memory
Rules of thumb

- Step 1: check if you can solve the problem “in that class”
- Step 2: if not, check if your problem “looks like” (is reducible from) the complete problem from the next class.

Of interest: PTIME-complete are not efficiently parallelizable.
Query complexity

Given a query $Q$ and a database $D$, what is the complexity of computing $Q(D)$?

- The answer depends on the query language:
  - Relational algebra, calculus, datalog

- Design tradeoff:
  - High complexity → rich queries
  - Low complexity → implemented efficiently
Complexity of query languages

Query Q, database D

- Data complexity
  - Fix Q, complexity $f(D)$
- Query complexity
  - Fix D, complexity $f(Q)$
- Combined complexity
  - Complexity $f(D, Q)$
Conventions

- Complexity is usually defined for a decision problem
  - We study the complexity of Boolean queries

- Complexity usually assumes some encoding of the input
  - We encode instances using binary representation
**Boolean queries**

**Definition**: A Boolean query is a query that returns either true or false.

---

**Non-Boolean**

```
SELECT DISTINCT R.x, S.y
FROM   R, S
WHERE R.z = S.z
```

```
Q(x,y) :- R(x,z), S(z,y)
T(x,y) :- R(x,y)
T(x,y) :- T(x,z), R(z,y)
```

**Boolean**

```
SELECT DISTINCT ‘yes’
FROM   R, S
WHERE R.x = ‘a’ and R.z = S.z
     and S.y = ‘b’
```

```
Q :- R(x,z), S(z,y)
T(x,y) :- R(x,y)
T(x,y) :- T(x,z), R(z,y)
Answer() :- T(‘a’,’b’)```
**Database encoding**

- **Encode** $\mathbf{D} = (D, R_1^D, \cdots, R_k^D)$ as follows:
  - Let $n = |\text{ADom}(D)|$
  - If $R_i$ has arity $k$, encode it as a string of $n^k$ bits:
    - 0 means element $(a_1, \cdots a_k) \notin R_i^D$
    - 1 means element $(a_1, \cdots a_k) \in R_i^D$

---

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>c</td>
</tr>
<tr>
<td>b</td>
<td>b</td>
</tr>
<tr>
<td>b</td>
<td>c</td>
</tr>
<tr>
<td>c</td>
<td>a</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>c</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

---

**Example**

<table>
<thead>
<tr>
<th>column 1</th>
<th>column 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>a</td>
</tr>
<tr>
<td>b</td>
<td>b</td>
</tr>
<tr>
<td>c</td>
<td>c</td>
</tr>
</tbody>
</table>

- 0 means element $(a, b, c) \notin R_i^D$
- 1 means element $(a, b, c) \in R_i^D$
Fix a Boolean query $Q$ in the query language. Determine the complexity of the following problem:

Given an input database instance $D = (D, R^D_1, \ldots, R^D_k)$, check if $Q(D) = \text{true}$.

This is also known as model checking problem: check if $D$ is a model for $Q$. 
What is the complexity of relational queries?

AC^0, L, NL, NC, PTIME, PSPACE, all computable
Example

\[ Q = \exists z. R(\text{'a',}z) \land S(z,\text{'b'}) \]

Prove that \( Q \) is in \( \text{AC}^0 \)

\[
R: \begin{array}{ccc}
  a & b & c \\
  \hline 
  a & 0 & 1 & 1 \\
  b & 0 & 1 & 1 \\
  c & 1 & 0 & 0 \\
\end{array}
\]

\[
S: \begin{array}{ccc}
  a & b & c \\
  \hline 
  a & 0 & 0 & 1 \\
  b & 1 & 1 & 0 \\
  c & 1 & 0 & 1 \\
\end{array}
\]
Example

Q = \exists z. R('a',z) \land S(z,'b')

Prove that Q is in AC^0

Circuit of depth 2

R:

<table>
<thead>
<tr>
<th></th>
<th>a</th>
<th>b</th>
<th>c</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>b</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>c</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

S:

<table>
<thead>
<tr>
<th></th>
<th>a</th>
<th>b</th>
<th>c</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>b</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>c</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

OR has n inputs
Each AND has 2 inputs
What is the complexity of relational queries?

All relational queries (expressible in RC) are in $\mathsf{AC}^0$. 

Complexity classes: $\mathsf{AC}^0$, $\mathsf{L}$, $\mathsf{NL}$, $\mathsf{NC}$, $\mathsf{PTIME}$, $\mathsf{PSPACE}$, all computable.
What is the complexity of datalog queries?

\[ T(x,y) :- R(x,y) \]
\[ T(x,y) :- T(x,z), R(z,y) \]
\[ \text{Answer()} :- T('a','b') \]
Datalog is not in $AC^0$

- Parity is not in $AC^0$
- We will reduce parity to the reachability problem
- Given input $(x_1, x_2, x_3, x_4, x_5) = (0, 1, 1, 0, 1)$
  construct the graph:

$$
\begin{align*}
T(x,y) &\ :- \ R(x,y) \\
T(x,y) &\ :- \ T(x,z), \ R(z,y) \\
\text{Answer}() &\ :- \ T('a_1','b_6')
\end{align*}
$$

The # of 1s is odd iff Answer is true
Datalog is in PTIME

- Fix any Boolean datalog program P.
- Given D, check if $P(D) = \text{true}$ is in PTIME

Proof argument:
- If an IDB has arity $k$, then it will reach its fixpoint in at most $n^k$ iterations.
Conjunctive Queries (CQ)

- A subset of FO (first order)
  - Less expressive

- Many queries in practice are conjunctive

- Some optimizers only handle CQs
  - Break larger queries into many CQs

- CQs have “better” theoretical properties than arbitrary queries
Conjunctive Queries

- \( R \): Extensional database (EDB) – stored
- \( P \): Intentional database (IDB) – computed

\[
P(x,z) :- R(x,y) \& R(y,z)
\]
Conjunctive Queries

- When facts in the body are true, we infer the head.
- Consider all possible assignments of variables in the body.

Example:

\[ P(x,z) : - R(x,y) \land R(y,z) \]
Conjunctive Queries

- A single datalog rule
- Equivalent to SELECT-DISTINCT-FROM-WHERE
- Select/project/join in RA
- Existential/conjunctive fraction of RC

Strictly speaking, we are not allowed to have non-equality selection predicates
Example

Find all employees having the same manager as ‘Smith’

\[ A(x) \text{ :- } \text{ManagedBy('Smith',y)} \& \text{ManagedBy(x,y)} \]

```
SELECT DISTINCT m2.name
FROM ManagedBy m1, ManagedBy m2
WHERE m1.name = 'Smith' AND
    m1.manager = m2.manager
```
Properties of CQ

**Satisfiability**
- A query is satisfiable if there exists some input relation R such that \( q(R) \) is non-empty
- **Every CQ is satisfiable**

**Monotonicity**
- A query is monotonic if for each instance I, J over the schema, \( I \subseteq J \) implies \( q(I) \subseteq q(J) \)
- **Every CQ is monotonic**
Satisfiability

We can always generate satisfying EDB relations from the body of the rule

\[ S(x,y,z) :\neg P(x,w) \& R(w,y,v) \& P(v,z) \]

<table>
<thead>
<tr>
<th>S</th>
<th>a</th>
<th>c</th>
<th>e</th>
</tr>
</thead>
<tbody>
<tr>
<td>P</td>
<td>a</td>
<td>b</td>
<td></td>
</tr>
<tr>
<td></td>
<td>d</td>
<td>e</td>
<td></td>
</tr>
<tr>
<td>R</td>
<td>b</td>
<td>c</td>
<td>d</td>
</tr>
</tbody>
</table>
Monotonicity

\[ \text{ans}(u):= R_1(u_1) \& \ldots \& R_n(u_n) \]

Consider 2 databases I, J, s.t. \( I \subseteq J \)

Let \( t \in q(I) \)

\[ \begin{align*}
\text{For some substitution } v : \\
& v(u_i) \in I(R_i) \text{ for each } i. \\
& t = v(u) \\
& \text{Since } I \subseteq J, \ v(u_i) \in J(R_i) \text{ for each } i. \\
& \text{So } t \in q(J)
\end{align*} \]
Consequence of monotonicity

This query is not monotone
Therefore, not CQ
It cannot be expressed as a simple SFW query

Q: Find all companies that make only products with price < 100!

```
SELECT DISTINCT C.cname
FROM Company C
WHERE 100 > ALL (SELECT price
                  FROM Product P
                  WHERE P.cid = C.cid)
```
Equivalence and containment

Needed for a variety of static analysis tasks
  Query optimization
  Query rewriting using views
  Testing for semijoin reductions
Query equivalence

**Definition:** Queries q1 and q2 are equivalent if for every database D, q1(D) = q2(D)

Notation: \( q1 \equiv q2 \)
Query containment

**Definition:** Query q1 is contained in q2 if for every database D, q1(D) ⊆ q2(D)

Notation: \( q_1 \subseteq q_2 \)

**Fact:** \( q_1 \subseteq q_2 \) and \( q_2 \subseteq q_1 \) iff \( q_1 \equiv q_2 \)

For the case of Boolean queries, containment is logical implication
Examples

Is $q_1 \subseteq q_2$? Yes

$q_1(x) :\dashv R(x,u), R(u,'Smith')$
$q_2(x) :\dashv R(x,u), R(u,v)$
Examples

Is $q_1 \subseteq q_2$? Yes

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

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

Is \( q_1 \subseteq q_2 \) ?  No

\begin{align*}
q_1(x) & : \text{R}(x,u), \text{R}(u,v), \text{R}(v,x) \\
q_2(x) & : \text{R}(x,u), \text{R}(u,x)
\end{align*}
Examples

Is $q_1 \subseteq q_2$ ?  \textbf{Yes}

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

Is $q_1 \subseteq q_2$? Yes

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

Is $q_1 \subseteq q_2$? Yes

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

**Theorem:** The query containment and query equivalence problems for CQ are NP-complete.

**Theorem:** The query containment and query equivalence problems for Relational Calculus are undecidable.
Query containment for CQ

Two ways to test

- Check if $q_2$ holds (produces the canonical tuple of $q_1$) on the canonical database of $q_1$
- Check if there exists a homomorphism $q_2 \to q_1$
Canonical database

- Canonical database for q1 is \( D = (D, R_1^D, \ldots, R_k^D) \)
  - \( D \): all variables and constants in q1
  - \( R_1^D, \ldots, R_k^D \): the body of q1

- Canonical tuple \( t_{q1} \) is the head of q1
Example

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

\[ \text{Canonical database } \mathbf{D} = (D, R^D) \]

\[ D = \{x,y,u,v\} \]

\[ R^D = \begin{array}{c|c}
  x & u \\
  \hline
  v & u \\
  v & y \\
\end{array} \]

\[ \text{ Canonical tuple } \quad t_{q_1} = (x,y) \]
Example

Canonical database $\mathbf{D} = (D, R)$

- $D = \{x, u, 'Smith', 'Fred'\}$
- $R = \{(x, u), (u, 'Smith'), (u, 'Fred'), (u, u)\}$

Canonical tuple $t_{q1} = (x)$
Checking containment using the canonical database

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

$D = \{x, y, u, v\}$

$R =$

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>u</td>
</tr>
<tr>
<td>v</td>
<td>u</td>
</tr>
<tr>
<td>v</td>
<td>y</td>
</tr>
</tbody>
</table>
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:

$f(\text{body}(q_2)) \subseteq \text{body}(q_1)$

$f(t_{q_1}) = t_{q_2}$
Example

\[ \text{var}(q1) = \{x, u, v, y\} \]
\[ \text{var}(q2) = \{x, u, v, w, t, y\} \]

\[ q1(x, y) : - R(x, u), R(v, u), R(v, y) \]
\[ q2(x, y) : - R(x, u), R(v, u), R(v, w), R(t, w), R(t, y) \]

q1 is contained in q2
Example

var(q1) U const(q1) = \{x,u,’Smith’\}

var(q2) = \{x,u,v,w\}

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

q2(x) :- R(x,u), R(u,v), R(u,’Smith’), R(w,u)

q1 is contained in q2
Complexity

**Theorem:** Checking containment of two CQs is NP-complete.

Φ = (¬X₃ ∨ ¬X₁ ∨ X₄) ∧ (X₁ ∨ X₂ ∨ X₃) ∧ (¬X₂ ∨ ¬X₃ ∨ X₁)

**Proof:** Reduction from 3-SAT

Given a 3CNF Φ

Step 1:
- construct q₁ independently of Φ

Step 2:
- construct q₂ from Φ

Prove:
- there exists a homomorphism q₂ → q₁ iff Φ is satisfiable
Proof: Step 1

There are 4 types of clauses in every 3SAT:
Type 1: $\neg X \lor \neg Y \lor \neg Z$
Type 2: $\neg X \lor \neg Y \lor Z$
Type 3: $\neg X \lor Y \lor Z$
Type 4: $X \lor Y \lor Z$

For each type, $q_1$ contains one relation with all 7 satisfying assignments, where $u=0$, $v=1$
Proof: Step 2

Constructing q2

q2 has one atom for each clause is Φ:

- Relation name is R1, or R2, or R3, or R4
- The variables are the same as those in the clause

Example:

\[ Φ = (\neg X_3 \lor \neg X_1 \lor X_4) \land (X_1 \lor X_2 \lor X_3) \land (\neg X_2 \lor \neg X_3 \lor X_1) \]

\[ q2 = R2(x_3,x_1,x_4), R4(x_1,x_2,x_3), R2(x_2,x_3,x_1) \]
Proof

Suppose there is a satisfying assignment for $\Phi$: it maps each $X_i$ to either 0 or 1

Define function $f: \text{Vars}(q2) \rightarrow \text{Vars}(q1)$:

- If $X_i = 0$ then $f(x_i) = u$
- If $X_i = 1$ then $f(x_i) = v$

Then $f$ is a homomorphism $f: q2 \rightarrow q1$

Suppose there exists a homomorphism $f: q2 \rightarrow q1$

Define the assignment:

- If $f(x_i) = u$ then $X_i = 0$
- If $f(x_i) = v$ then $X_i = 1$

This is a satisfying assignment for $\Phi$
Beyond CQ

- Containment for arbitrary relational queries is undecidable
- Any static analysis on relational queries is undecidable
- All these results follow from Trakhtenbrot’s theorem
Query containment for UCQ

\[ q_1 \cup q_2 \cup q_3 \ldots \subseteq q'_1 \cup q'_2 \cup q'_3 \ldots \]

Note:
\[ q_1 \cup q_2 \cup q_3 \ldots \subseteq q \quad \text{iff} \quad q_1 \subseteq q \quad \text{and} \quad q_2 \subseteq q \quad \text{and} \quad \ldots \]

**Theorem:** \[ q \subseteq q'_1 \cup q'_2 \cup q'_3 \ldots \quad \text{iff there exists some} \ k \ \text{such that} \quad q \subseteq q'_k \]
Definition: A conjunctive query $q$ is minimal, if for every other query $q'$ such that $q \equiv q'$, $q'$ 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

Algorithm:
- Choose a subgoal $g$ of $q$
- Remove $g$: let $q'$ be the new query
- $q \subseteq q'$ ← Why?
- If $q' \subseteq q$, then permanently remove $g$

The order in which we inspect subgoals doesn't matter
In practice

- No database system performs minimization
  - It’s hard
  - Users usually write minimal queries

- Non-minimal queries arise when using views intensely
Remember Semijoins?

\[ R \bowtie S = \Pi_{A_1, \ldots, A_n} (R \bowtie S) \]

\[ R \bowtie S = (R \bowtie S) \bowtie S \]

Prove that the following two datalog queries are equivalent:

q1(x,y,z) :- R(x,y), S(x,z)

R1(x,y) :- R(x,y), S(x,z)
q2(x,y,z) :- R1(x,y), S(x,z)

q1(x,y,z) :- R(x,y), S(x,z)
q2(x,y,z) :- R(x,y), S(x,u), S(x,z)

q1(x,y,z) :- R(x,y), S(x,z)
q2(x,y,z) :- R(x,y), S(x,u), S(x,z)
Semijoins

- Important in distributed databases
- Often combined with Bloom filters
- See 22.10.2 in the textbook
Semijoin Reducer

Given a query: \( Q = R_1 \bowtie R_2 \bowtie \cdots \bowtie R_n \)

A semijoin reducer for \( q \) is:
\[
\begin{align*}
R_{i1} &= R_{i1} \bowtie R_{j1} \\
R_{i2} &= R_{i2} \bowtie R_{j2} \\
& \quad \vdots \\
R_{ip} &= R_{ip} \bowtie R_{jp}
\end{align*}
\]

Such that the query is equivalent to
\[
Q = R_{k1} \bowtie R_{k2} \bowtie \cdots \bowtie R_{kn}
\]

In a full reducer, no dangling tuples remain
Example

Q = R(A,B) ⋈ S(B,C)

Semijoin reducer:
R1(A,B) = R(A,B) ⋈ S(B,C)

Re-written query: Q = R1(A,B) ⋈ S(B,C)

Are there any dangling tuples?
Example

\[ Q = \text{R}(A,B) \bowtie \text{S}(B,C) \]

\[
\text{Full semijoin reducer:} \\
\quad \text{R}_1(A,B) = \text{R}(A,B) \bowtie \text{S}(B,C) \\
\quad \text{S}_1(B,C) = \text{S}(B,C) \bowtie \text{R}_1(A,B)
\]

\[
\text{Re-written query: } Q = \text{R}_1(A,B) \bowtie \text{S}_1(B,C)
\]

No more dangling tuples
Example

More complex:
\[ Q = R(A,B) \Join S(B,C) \Join T(C,D,E) \]

Full reducer:
\[ S'(B,C) = S(B,C) \Join R(A,B) \]
\[ T'(C,D,E) = T(C,D,E) \Join S'(B,C) \]
\[ S''(B,C) = S'(B,C) \Join T'(C,D,E) \]
\[ R'(A,B) = R(A,B) \Join S''(B,C) \]

\[ Q = R'(A,B) \Join S''(B,C) \Join T'(C,D,E) \]
Semijoin Reducer

Example: $Q = R(A,B) \bowtie S(B,C) \bowtie T(C,A)$

No full reducer

Theorem: A query has a full reducer iff it is “acyclic”.
Expressive power of FO

Let $R(x,y)$ represent a graph

Query $\text{path}(x,y) =$

All $x,y$ such that there is a path from $x$ to $y$

Theorem: $\text{path}(x,y)$ cannot be expressed in FO
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 into:
- $A(x,y) :- R(x,u), R(u,v), R(v,w), R(w,m), R(m,n), R(n,y)$
Non-recursive datalog with negation

- Expresses FO queries
  - Negated subgoals
  - Implicit union

- Can evaluate in an order such that all body predicates have been evaluated.
Recursion

Two forms of transitive closure

Path(x,y) :- R(x,y)
Path(x,y) :- Path(x,u), R(u,y)

Path(x,y) :- R(x,y)
Path(x,y) :- Path(x,u), Path(u,y)
Recursion example

- EDB \( \text{Par}(c,p) = p \) is parent of \( c \)

- Generalized cousins: people with common ancestors one or more generations back

\[
\begin{align*}
\text{Sib}(x,y) & : \text{Par}(x,p), \text{Par}(y,p), x \neq y \\
\text{Cousin}(x,y) & : \text{Sib}(x,y) \\
\text{Cousin}(x,y) & : \text{Par}(x,xp), \text{Par}(y,yp), \text{Cousin}(xp,yp)
\end{align*}
\]
Definition of recursion

- Form a dependency graph whose nodes are IDB predicates
- Connect $X \rightarrow Y$ iff there is a rule with $X$ in the head and $Y$ in the body

- Cycle = recursion; no cycle = no recursion
Meaning of datalog rules

Model-theoretic
- Rules define a set of satisfying relations
- Whenever body is true, head is true

Proof-theoretic
- Set of facts derivable from EDB relations by applying the rules.
Evaluating recursive rules

- This works if there is no negation
  - Start with all IDB relations empty
  - Repeatedly evaluate the rules using the EDB and the previous IDB to get the new IDB
  - End when there is no change in the IDB relations
“Naïve” evaluation algorithm

Start: IDB = ∅

Apply rules to IDB, EDB

yes

Change to IDB?

no

done
Semi-naïve evaluation

- Since the EDB never changes, on each round we only get new IDB tuples if we use at least one IDB tuple obtained in the previous round.

- Saves work; lets us avoid re-discovering most known facts.
  - Though a fact can still be derived in more than one way.
Par data: parent above child

Round 1

Round 2

Round 3
Recursion + negation

- Naïve evaluation doesn’t work when there are negated subgoals

- Negation wrapped in recursion makes no sense in general

- Even when they are separate, we can have ambiguity about the correct IDB relations
Stratified negation

- Stratification is a constraint usually placed on datalog with recursion and negation.
- It rules out negation wrapped inside recursion.
Example

Suppose \( R = \{ (1) \} \)

Two models satisfy the rules:

\[
\begin{align*}
P(x) & : -R(x) \& \neg Q(x) \\
Q(x) & : -R(x) \& \neg P(x)
\end{align*}
\]

\[
\begin{align*}
P = \{ \} , Q = \{ 1 \} \\
P = \{ 1 \} , Q = \{ \} 
\end{align*}
\]
Intuitively, the stratum of an IDB predicate $P$ is the maximum number of negations that can be applied to an IDB predicate used in evaluating $P$.

Stratified negation $= \text{finite strata}$
Stratum graph

- Nodes = IDB predicates
- Connect $A \rightarrow B$ if predicate $A$ depends on $B$
- Label the edge “-” if the $B$ subgoal is negated

- The stratum is the maximum number of “-” edges on a path leading from that node
- A datalog program is stratified if all its IDB predicates have finite strata
Example

\[
\begin{align*}
  P(x) & : - R(x) \& \neg Q(x) \\
  Q(x) & : - R(x) \& \neg P(x)
\end{align*}
\]
The stratified model

- When a datalog program is stratified, we can evaluate IDB predicates lowest-stratum-first.

- Once evaluated, treat it as EDB for higher strata.
Summary

- Query complexity
- Conjunctive queries
- Containment, equivalence, minimality
- Semijoin reductions
- Recursive datalog