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

- **AC⁰**
  - 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 $\rightarrow$ rich queries
- Low complexity $\rightarrow$ 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 $f(D,Q)$

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

<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></th>
<th>0</th>
<th>1</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>
Data complexity

- Fix a Boolean query $Q$ in the query language. Determine the complexity of the following problem:
  - Given an input database instance $D = (D, R_1^D, \ldots, R_k^D)$ 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?
Example

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

Prove that \( Q \) is in \( AC^0 \)

<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>

<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>
Example

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

Prove that Q is in AC\(^0\)

Circuit of depth 2

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

<table>
<thead>
<tr>
<th>S: a</th>
<th>b</th>
<th>c</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>b</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>c</td>
<td>1</td>
<td>0</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 $\text{AC}^0$
What is the complexity of datalog queries?

\[
T(x,y) :- R(x,y) \\
T(x,y) :- T(x,z), R(z,y) \\
Answer() :- T(‘a’, ’b’) \\
\]
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 $P(D) = true$ 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

**Formula**: $P(x,z) : \neg R(x,y) \& R(y,z)$

- **if**
- **variables**
- **subgoals**
- **conjunction**
- **head**
- **body**
- **implicit $\exists$**
Conjunctive Queries

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

```
P(x,z) :- R(x,y) & 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’

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

\[ \text{SELECT DISTINCT m2.name} \]
\[ \text{FROM ManagedBy m1, ManagedBy m2} \]
\[ \text{WHERE m1.name = ‘Smith’ AND} \]
\[ \quad \text{m1.manager = m2.manager} \]
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

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

\[
\begin{array}{ccc}
S & P & R \\
a & c & e \\
& a & b \\
& d & e \\
b & c & d \\
d & e \\
\end{array}
\]
Monotonicity

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

- Consider 2 databases \( I, J \), s.t. \( I \subseteq J \)
- Let \( t \in q(I) \)
  - For some substitution \( v \):
    - \( v(u_i) \in I(R_i) \) for each \( i \).
    - \( t = v(u) \)
  - Since \( I \subseteq J \), \( v(u_i) \in J(R_i) \) for each \( i \).
  - So \( t \in q(J) \)
Consequence of monotonicity

Product (pname, price, cid)
Company (cid, cname, city)

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)
```

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

- Needed for a variety of static analysis tasks
  - Query optimization
  - Query rewriting using views
  - Testing for semijoin reductions
**Definition:** 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**

**Definition:** Queries $q_1$ is contained in $q_2$ if for every database $D$, $q_1(D) \subseteq q_2(D)$

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

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

Is $q_1 \subseteq q_2$?  Yes

$q_1(x) :\text{-} R(x,u), R(u,v), R(v,w)$
$q_2(x) :\text{-} R(x,u), R(u,v)$
Examples

Is $q_1 \subseteq q_2$? No

$q_1(x) :\neg R(x,u), R(u,v), R(v,x)$
$q_2(x) :\neg R(x,u), R(u,x)$
Examples

Is \( q_1 \subseteq q_2 \) ? Yes

\[
q_1(x) :- R(x,u), R(u,y) \\
q_2(x) :- R(x,u), R(v,u), R(u,y)
\]
Examples

Is \( q_1 \subseteq q_2 \) ? \textbf{Yes}

\[
q_1(x) : \text{ R}(x,u), \text{ R}(u,v) \\
q_2(x) : \text{ R}(x,u), \text{ R}(x,y), \text{ R}(u,v), \text{ R}(u,w)
\]
Examples

Is \( q_1 \subseteq q_2 \)?  Yes

\[
q_1(x) :\text{ } R(x,u), \text{ } R(u,u)
\]

\[
q_2(x) :\text{ } R(x,u), \text{ } R(u,v), \text{ } R(v,w)
\]
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 q2 holds on the canonical database of q1
  - Check if there exists a homomorphism $q_2 \rightarrow q_1$
 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

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

- Canonical database $D = (D, R^D)$
  - $D = \{x, y, u, v\}$
  - $R^D = \begin{array}{cc}
    x & u \\
    v & u \\
    v & y \\
  \end{array}$
  - Canonical tuple $t_{q1} = (x, y)$
Example

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

- Canonical database $\mathbf{D} = (D, R)$
  - $D = \{x,u,’Smith’,’Fred’\}$
  - $R = \begin{array}{c|c}
    x & u \\
    \hline
    u & ‘Smith’ \\
    \hline
    u & ‘Fred’ \\
    \hline
    u & u
  \end{array}$

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

D={x, y, u, v}

R =

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

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
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

\[
\begin{align*}
\text{var}(q1) \cup \text{const}(q1) &= \{x,u,'Smith'\} \\
\text{var}(q2) &= \{x,u,v,w\}
\end{align*}
\]

\[
\begin{align*}
q1(x) &:\ R(x,u), \ R(u,'Smith'), \ R(u,'Fred'), \ R(u,u) \\
q2(x) &:\ R(x,u), \ R(u,v), \ R(u,'Smith'), \ R(w,u)
\end{align*}
\]

q1 is contained in q2
Complexity

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

\[ \Phi = (\neg X_3 \vee \neg X_1 \vee X_4) \land (X_1 \vee X_2 \vee X_3) \land (\neg X_2 \vee \neg X_3 \vee X_1) \]

**Proof:** Reduction from 3-SAT

Given a 3CNF \( \Phi \)

**Step 1:**

construct \( q_1 \) independently of \( \Phi \)

**Step 2:**

construct \( q_2 \) from \( \Phi \)

**Prove:**

there exists a homomorphism \( q_2 \rightarrow q_1 \) iff \( \Phi \) 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, q1 contains one relation with all 7 satisfying assignments, where $u=0$, $v=1$

<table>
<thead>
<tr>
<th>R1 (misses $v,v,v$)</th>
<th>R2 (misses $v,v,u$)</th>
<th>R3 (misses $v,u,u$)</th>
<th>R4 (misses $u,u,u$)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$u \ u \ u$</td>
<td>$u \ u \ u$</td>
<td>$u \ u \ u$</td>
<td>$u \ u \ v$</td>
</tr>
<tr>
<td>$u \ u \ v$</td>
<td>$u \ u \ v$</td>
<td>$u \ u \ v$</td>
<td>$u \ v \ u$</td>
</tr>
<tr>
<td>$u \ v \ u$</td>
<td>$u \ v \ u$</td>
<td>$u \ v \ u$</td>
<td>$u \ v \ v$</td>
</tr>
<tr>
<td>$u \ v \ v$</td>
<td>$u \ v \ v$</td>
<td>$u \ v \ v$</td>
<td>$v \ u \ u$</td>
</tr>
<tr>
<td>$v \ u \ u$</td>
<td>$v \ u \ u$</td>
<td>$v \ u \ v$</td>
<td>$v \ u \ v$</td>
</tr>
<tr>
<td>$v \ u \ v$</td>
<td>$v \ u \ v$</td>
<td>$v \ v \ u$</td>
<td>$v \ v \ u$</td>
</tr>
<tr>
<td>$v \ v \ u$</td>
<td>$v \ v \ v$</td>
<td>$v \ v \ v$</td>
<td>$v \ v \ v$</td>
</tr>
</tbody>
</table>
Proof: Step 2

Constructing q2

q2 has one atom for each clause is $\Phi$:
- Relation name is R1, or R2, or R3, or R4
- The variables are the same as those in the clause

Example:

$\Phi = (\neg X_3 \lor \neg X_1 \lor X_4) \land (X_1 \lor X_2 \lor X_3) \land (\neg X_2 \lor \neg X_3 \lor X_1)$

$q_2 = R2(x_3, x_1, x_4), R4(x_1, x_2, x_3), R2(x_2, x_3, x_1)$
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
Trakhtenbrot’s theorem

**Definition:** A sentence $\varphi$ is called **finitely satisfiable** if there exists a finite database instance $D$ s.t. $D \models \varphi$

**Theorem:** The following problem is undecidable:
Given FO sentence $\varphi$, check if $\varphi$ is finitely satisfiable.
Query containment for UCQ

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

Note:

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

**Theorem:** \[ q \subseteq q'_1 \cup q'_2 \cup q'_3 \ldots \quad \text{iff} \quad \text{there exists some} \; k \quad \text{such that} \quad q \subseteq q'_k \]
Query minimization

**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
- $q \subseteq q' \quad \text{Why?}$
- 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:

\[ q_1(x,y,z) :- R(x,y), S(x,z) \]

\[ q_2(x,y,z) :- R(x,y), S(x,u), S(x,z) \]

\[ R_1(x,y) :- R(x,y), S(x,z) \]

\[ q_2(x,y,z) :- R_1(x,y), S(x,z) \]

\[ q_1(x,y,z) :- R(x,y), S(x,z) \]

\[ q_2(x,y,z) :- R(x,y), S(x,u), S(x,z) \]
Semijoins

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

- Given a query: \( Q = R_1 \bowtie R_2 \bowtie ... \bowtie R_n \)
- A **semijoin reducer** for \( q \) is:
  - \( R_{i1} = R_{i1} \bowtie R_{j1} \)
  - \( R_{i2} = R_{i2} \bowtie R_{j2} \)
  - ...
  - \( R_{ip} = R_{ip} \bowtie R_{jp} \)
- Such that the query is equivalent to
  - \( Q = R_{k1} \bowtie R_{k2} \bowtie ... \bowtie R_{kn} \)
- In a **full reducer**, no dangling tuples remain
Example

- $Q = R(A,B) \bowtie S(B,C)$

- Semijoin reducer:
  
  $R_1(A,B) = R(A,B) \bowtie S(B,C)$

- Re-written query: $Q = R_1(A,B) \bowtie S(B,C)$

Are there any dangling tuples?
Example

- Q = R(A,B) ⋈ S(B,C)

- Full semijoin reducer:
  
  \[ R_1(A,B) = R(A,B) \bowtie S(B,C) \]
  \[ S_1(B,C) = S(B,C) \bowtie R_1(A,B) \]

- Re-written query: Q = R1(A,B) ⋈ S1(B,C)

No more dangling tuples
Example

- More complex: \( Q = R(A,B) \bowtie S(B,C) \bowtie T(C,D,E) \)
- Full reducer:
  \[
  S'(B,C) = S(B,C) \bowtie R(A,B) \\
  T'(C,D,E) = T(C,D,E) \bowtie S'(B,C) \\
  S''(B,C) = S'(B,C) \bowtie T'(C,D,E) \\
  R'(A,B) = R(A,B) \bowtie S''(B,C)
  \]

- \( Q = R'(A,B) \bowtie S''(B,C) \bowtie T'(C,D,E) \)
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

```
Sib(x, y) :- Par(x, p), Par(y, p), x ≠ y
Cousin(x, y) :- Sib(x, y)
Cousin(x, y) :- Par(x, xp), Par(y, yp), Cousin(xp, yp)
```
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

Change to IDB?

yes

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

\[
P(x) : \neg R(x) \land \neg Q(x) \\
Q(x) : \neg R(x) \land \neg P(x)
\]

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

- Two models satisfy the rules:
  - \( P = \{ \}, Q=\{1\} \)
  - \( P=\{1\}, Q=\{ \} \)
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 = 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

P(x) :- R(x) & \neg Q(x)
Q(x) :- R(x) & \neg P(x)

- Not stratified

\begin{tikzpicture}[->,>=stealth,thick,shorten >=1pt]
  \draw (0,0) -- (1,0) node [midway,left] {\texttt{P}};
  \draw (0,0) -- (0,1) node [midway,above] {\texttt{Q}};
\end{tikzpicture}
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