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^0
- L (LOGSPACE)
- NL (NLOGSPACE)
- 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.

- Complexity
- Equivalence
- Containment
- Minimization
- Semijoin reduction
- Datalog evaluation

Query complexity

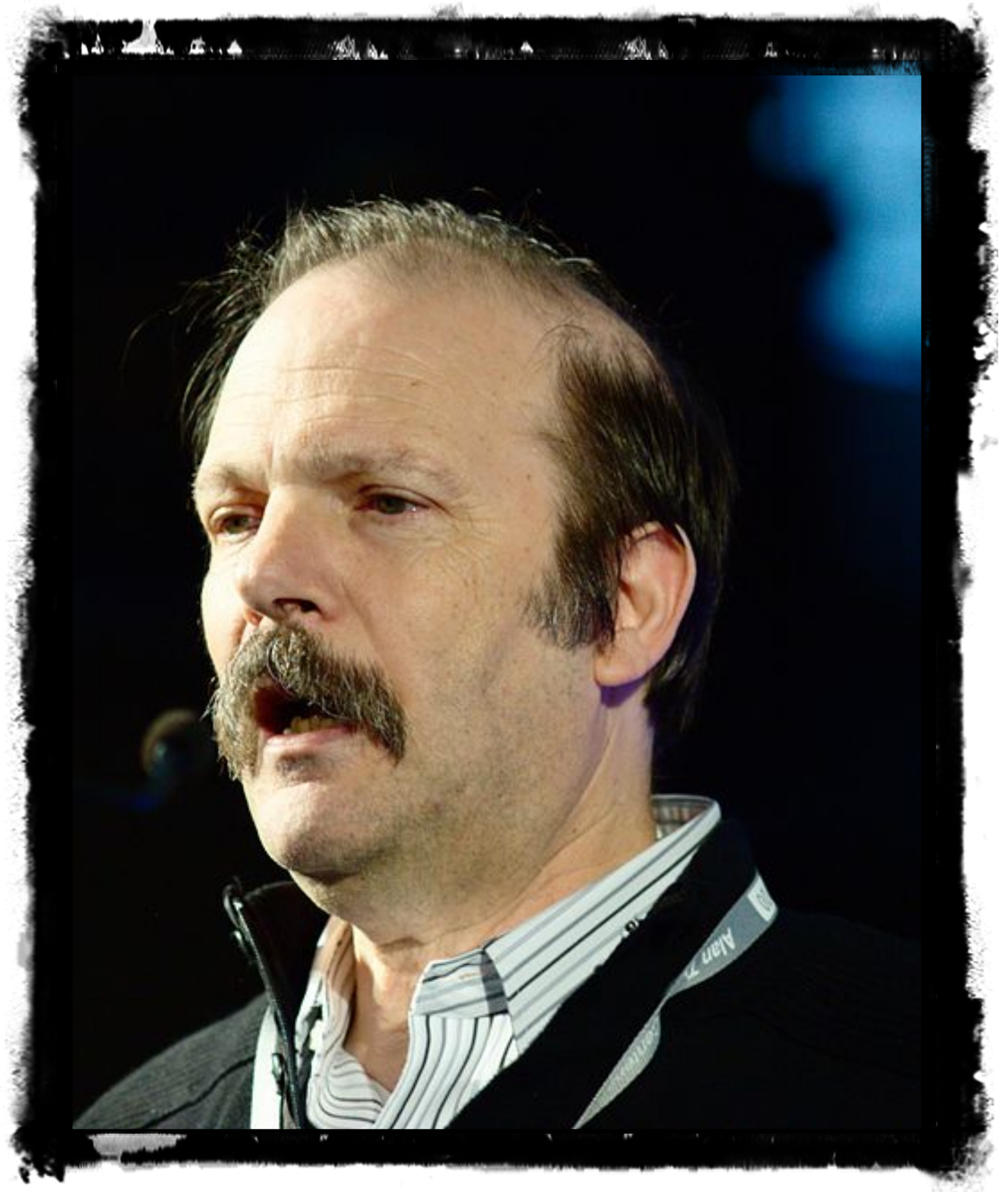
Given a query Q and a database D, what is the complexity of computing Q(D)?

- 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)



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 $\mathbf{D} = (D, R_1^D, \cdots, R_k^D)$ as follows:
  - Let $n = |ADom(D)|$
  - 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 $(a_1, \cdots a_k) \in R_i^D$

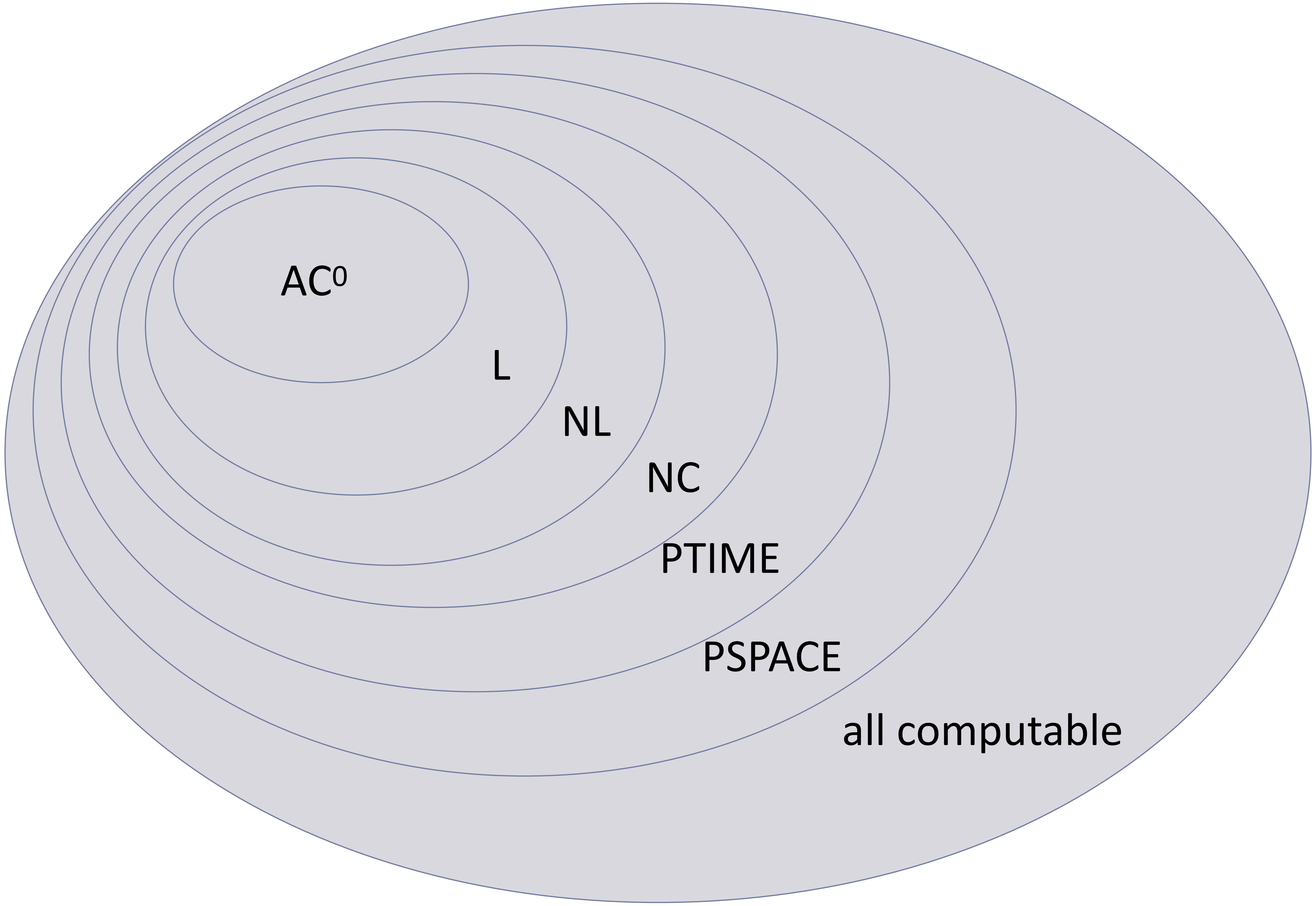
a	b
a	c
b	b
b	c
c	a

column 1 \ column 2	a	b	c
a	0	1	1
b	0	1	1
c	1	0	0

Data complexity

- Fix a Boolean query Q in the query language. Determine the complexity of the following problem:
  - Given an input database instance $\mathbf{D} = (D, R_1^D, \cdots, R_k^D)$ check if $Q(\mathbf{D}) = \text{true}$
  - This is also known as model checking problem: check if $\mathbf{D}$ 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

R:

	a	b	c
a	0	1	1
b	0	1	1
c	1	0	0

S:

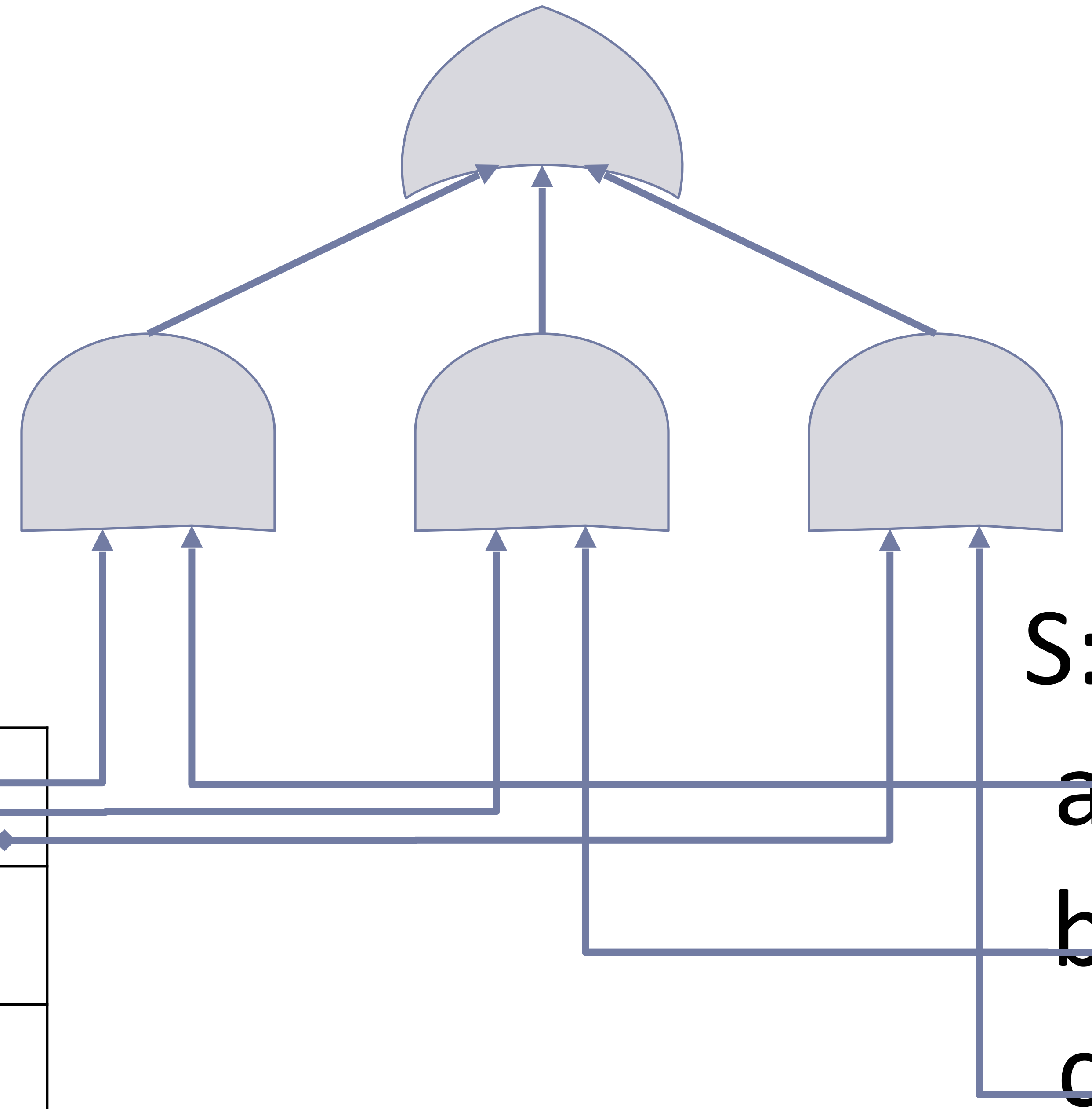
	a	b	c
a	0	0	1
b	1	1	0
c	1	0	1

Example

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

Prove that Q is in AC^0

Circuit of depth 2



OR has n inputs

Each AND has 2 inputs

R:

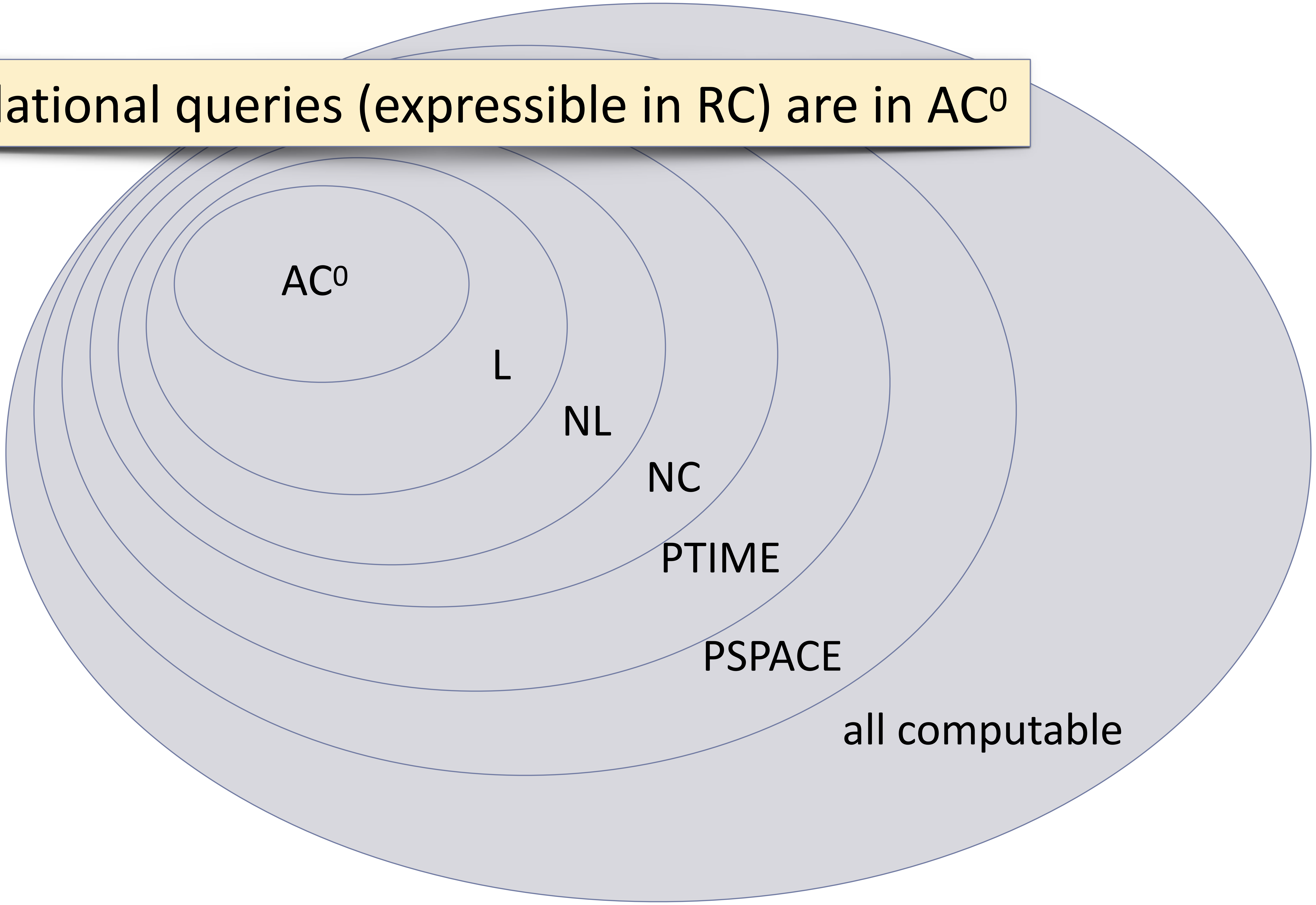
	a	b	c
a	0	1	1
b	0	1	1
c	1	0	0

S:

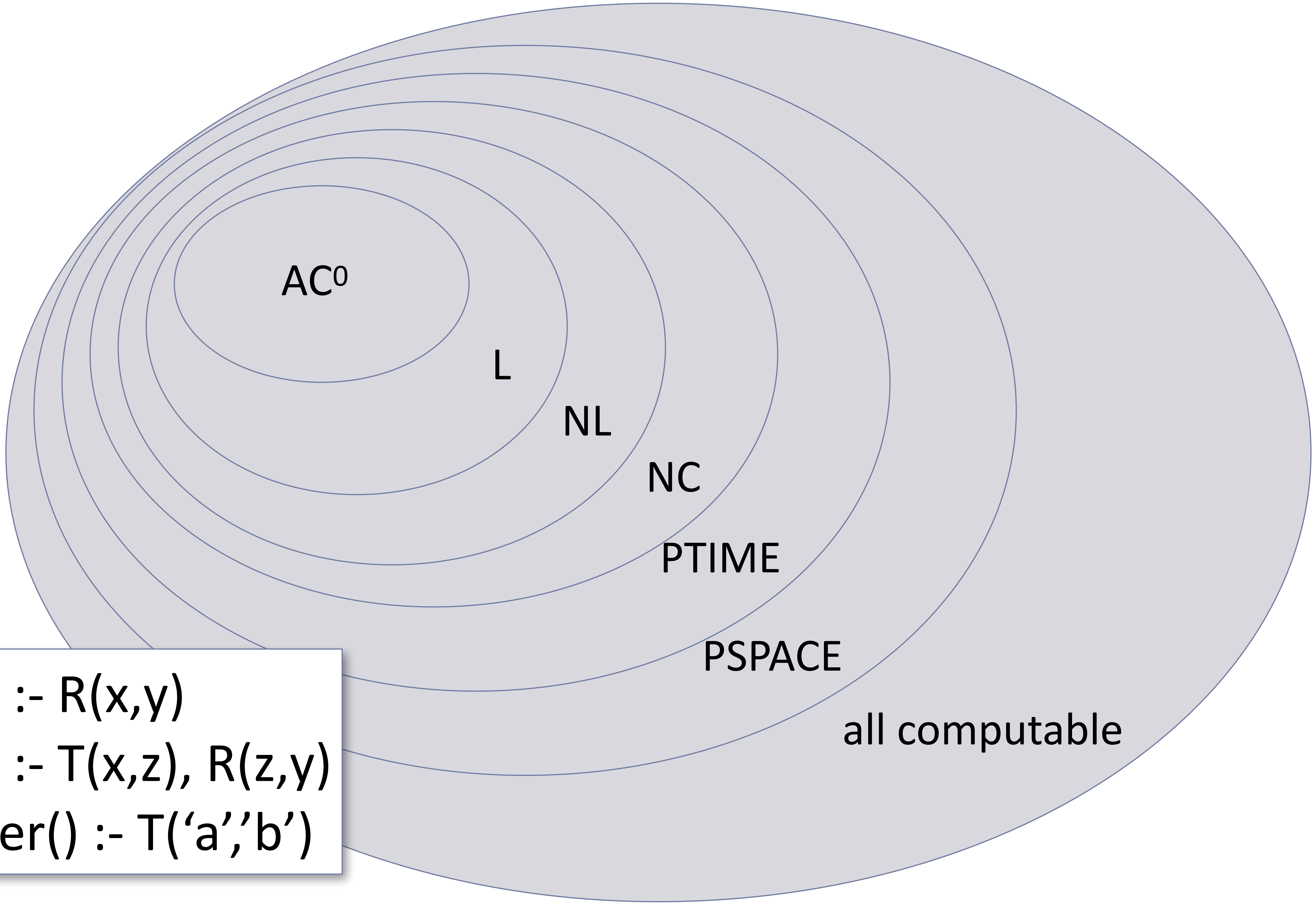
	a	b	c
a	0	0	1
b	1	1	0
c	1	0	1

What is the complexity of relational queries?

All relational queries (expressible in RC) are in AC^0



What is the complexity of datalog queries?

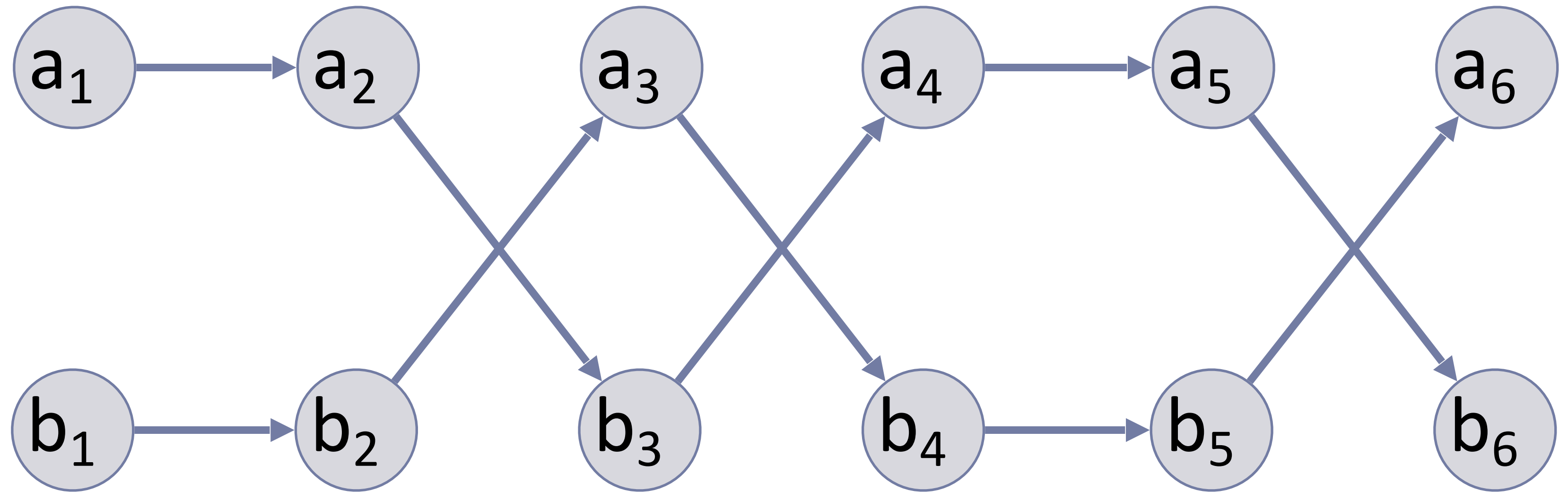



```
T(x,y) :- R(x,y)
T(x,y) :- T(x,z), R(z,y)
Answer() :- T('a','b')
```

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:

```
T(x,y) :- R(x,y)
T(x,y) :- T(x,z), R(z,y)
Answer() :- T('a1','b6')
```



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

variables

“if”

subgoals

$$\underbrace{P(x,z)}_{\text{head}} \text{ :- } \underbrace{R(x,y) \;\&\; R(y,z)}_{\text{body}}$$

implicit $\exists$

conjunction

Conjunctive Queries

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

variables

"if"

subgoals

$$\underbrace{P(x,z)}_{\text{head}}\ \text{:-}\ \underbrace{R(x,y)\ \&\ R(y,z)}_{\text{body}}$$

implicit ∃

conjunction

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) :- ManagedBy('Smith',y) & 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
  - <u>Every CQ is satisfiable</u>
- Monotonicity
  - A query is monotonic if for each instance I, J over the schema, $I \subseteq J$ implies $q(I) \subseteq q(J)$
  - <u>Every CQ is monotonic</u>

Satisfiability

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

S(x,y,z) :- P(x,w) & R(w,y,v) & P(v,z)

a b   b c d   d e

S

a	c	e

P

a	b
d	e

R

b	c	d

Monotonicity

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

- 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)

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

Q: Find all companies that make <u>only</u> 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 <u>equivalent</u> if for every database D, q1(D) = q2(D)

Notation: $q1 \equiv q2$

Query containment

Definition: Query q1 is <u>contained</u> in q2 if for every database D, q1(D)$\subseteq$q2(D)

Notation: q1 $\subseteq$ q2

Fact: q1 $\subseteq$ q2 and q2 $\subseteq$ q1 iff q1 $\equiv$ q2

For the case of Boolean queries, containment is logical implication

Examples

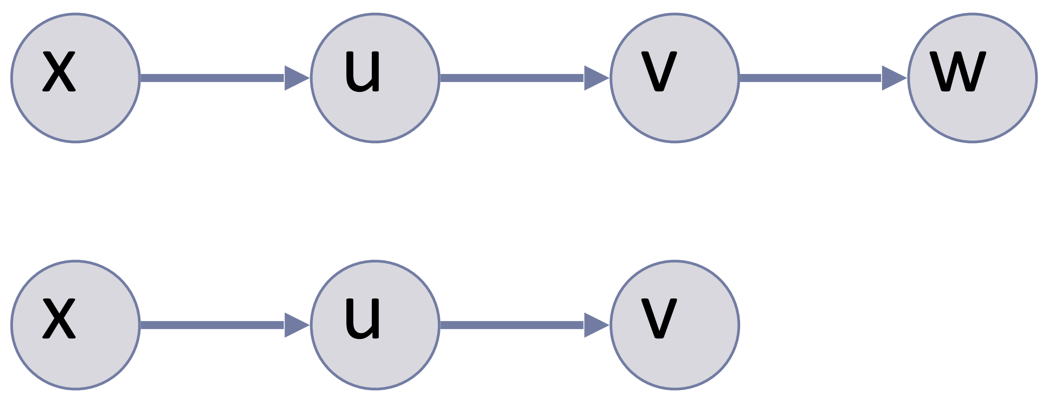
Is $q1 \subseteq q2$?   Yes

q1(x) :- R(x,u), R(u,'Smith')
q2(x) :- R(x,u), R(u,v)

Examples

Is q1 $\subseteq$ q2 ? Yes

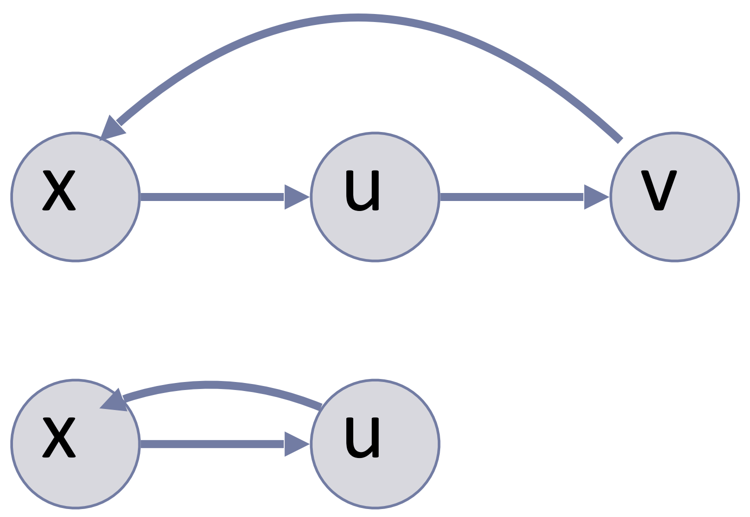
q1(x) :- R(x,u), R(u,v), R(v,w)
q2(x) :- R(x,u), R(u,v)



Examples

Is $q1 \subseteq q2$? No

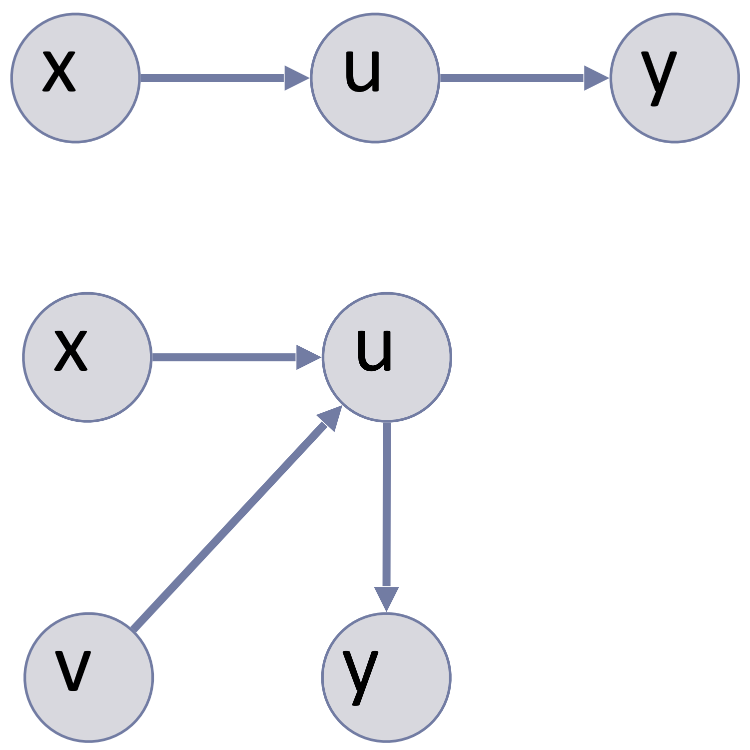
q1(x) :- R(x,u), R(u,v), R(v,x)
q2(x) :- R(x,u), R(u,x)



Examples

Is q1 $\subseteq$ q2 ?   Yes

q1(x) :- R(x,u), R(u,y)
q2(x) :- R(x,u), R(v,u), R(u,y)

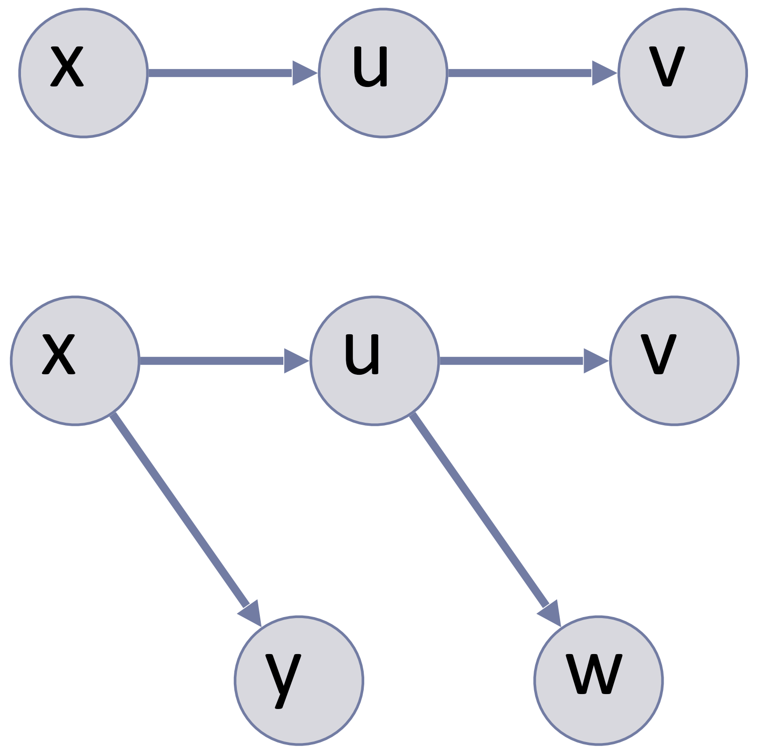


Examples

Is q1 $\subseteq$ q2 ?

Yes

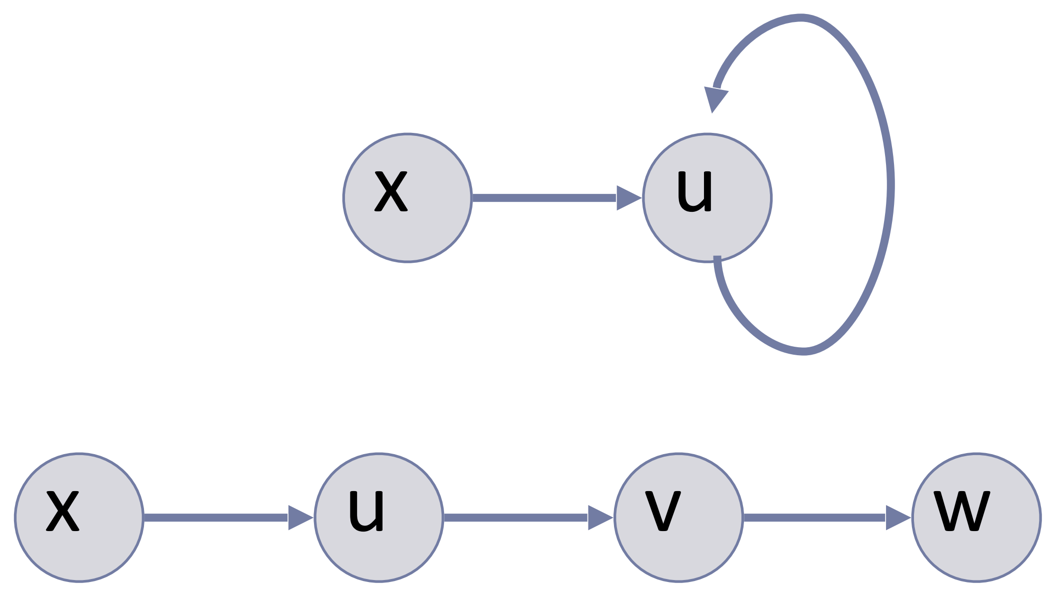
q1(x) :- R(x,u), R(u,v)
q2(x) :- R(x,u), R(x,y), R(u,v), R(u,w)



Examples

Is $q1 \subseteq q2$?   Yes

q1(x) :- R(x,u), R(u,u)
q2(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 q2 holds (produces the canonical tuple of q1) on the canonical database of q1
  - Check if there exists a homomorphism q2 → q1

Canonical database

- Canonical database for q1 is $\mathbf{D} = (D, R_1^D, \cdots, R_k^D)$
  - D: all variables and constants in q1
  - $R_1^D, \cdots, R_k^D$: 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)
  - $D = \{x,y,u,v\}$
  - $R^D =$

x	u
v	u
v	y

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

Example

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

- Canonical database **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

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)

- D={x,y,u,v}
- R =

x	u
v	u
v	y

q1 is contained in q2

Query homomorphisms

- A homomorphism f: q2 → q1 is a function f:var(q2) → var(q1) U const(q1), such that:
  - $f(body(q2)) \subseteq body(q1)$
  - $f(t_{q1}) = t_{q2}$

Example

var(q1) = {x,u,v,y}

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.

$$\Phi = (\neg X_3 \vee \neg X_1 \vee X_4) \wedge (X_1 \vee X_2 \vee X_3) \wedge (\neg X_2 \vee \neg X_3 \vee X_1)$$

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

Constructing q1

There are 4 types of clauses in every 3SAT:

Type 1: $\neg X \vee \neg Y \vee \neg Z$

Type 2: $\neg X \vee \neg Y \vee Z$

Type 3: $\neg X \vee Y \vee Z$

Type 4: $X \vee Y \vee Z$

For each type, q1 contains one relation with all 7 satisfying assignments, where u=0, v=1

R1
(misses v,v,v)

u	u	u
u	u	v
u	v	u
u	v	v
v	u	u
v	u	v
v	v	u

R2
(misses v,v,u)

u	u	u
u	u	v
u	v	u
u	v	v
v	u	u
v	u	v
v	v	v

R3
(misses v,u,u)

u	u	u
u	u	v
u	v	u
u	v	v
v	u	v
v	v	u
v	v	v

R4
(misses u,u,u)

u	u	v
u	v	u
u	v	v
v	u	u
v	u	v
v	v	u
v	v	v

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:

$$\Phi = (\neg X_3 \vee \neg X_1 \vee X_4) \wedge (X_1 \vee X_2 \vee X_3) \wedge (\neg X_2 \vee \neg X_3 \vee X_1)$$

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 Φ : it maps each X_i to either 0 or 1
  - Define function f: Vars(q2)→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→q1
- Suppose there exists a homomorphism f: q2→q1
  - Define the assignment:
    - If $f(x_i) = u$ then $X_i = 0$
    - If $f(xi) = v$ then $X_i = 1$
  - This is a satisfying assignment for Φ

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:

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

Theorem: $q \subseteq q'_1 \cup q'_2 \cup q'_3 \ldots$ iff there exists some k such that $q \subseteq q'_k$

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' ⟵ 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?

$$R \ltimes S = \Pi_{A1,\ldots,An} (R \bowtie S)$$

$$R \bowtie S = (R \ltimes S) \bowtie S$$

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: $Q = R1 \bowtie R2 \bowtie \cdots \bowtie Rn$
- A <u>semijoin reducer</u> for q is:
  - $R_{i1} = R_{i1} \ltimes R_{j1}$
  - $R_{i2} = R_{i2} \ltimes R_{j2}$
  - $\cdots$
  - $R_{ip} = R_{ip} \ltimes R_{jp}$
- Such that the query is equivalent to
  - $Q = R_{k1} \bowtie R_{k2} \bowtie \cdots \bowtie R_{kn}$
- In a <u>full reducer</u>, no dangling tuples remain

Example

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

- Semijoin reducer:

  $R1(A,B) = R(A,B) \ltimes S(B,C)$

- Re-written query: $Q = R1(A,B) \bowtie S(B,C)$

Are there any dangling tuples?

Example

- $Q = R(A,B) \bowtie S(B,C)$
- Full semijoin reducer:

  $R1(A,B) = R(A,B) \ltimes S(B,C)$

  $S1(B,C) = S(B,C) \ltimes R1(A,B)$
- Re-written query: $Q = R1(A,B) \bowtie S1(B,C)$

No more dangling tuples

Example

- More complex:
  $Q = R(A,B) \bowtie S(B,C) \bowtie T(C,D,E)$
- Full reducer:

$S'(B,C) = S(B,C) \ltimes R(A,B)$

$T'(C,D,E) = T(C,D,E) \ltimes S'(B,C)$

$S''(B,C) = S'(B,C) \ltimes T'(C,D,E)$

$R'(A,B) = R(A,B) \ltimes S''(B,C)$

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

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 path(x,y) =
  - All x,y such that there is a path from x to y

- Theorem: 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 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→Y iff there is a rule with X in the head and Y in the body
- Cycle = recursion; no cycle = no recursion

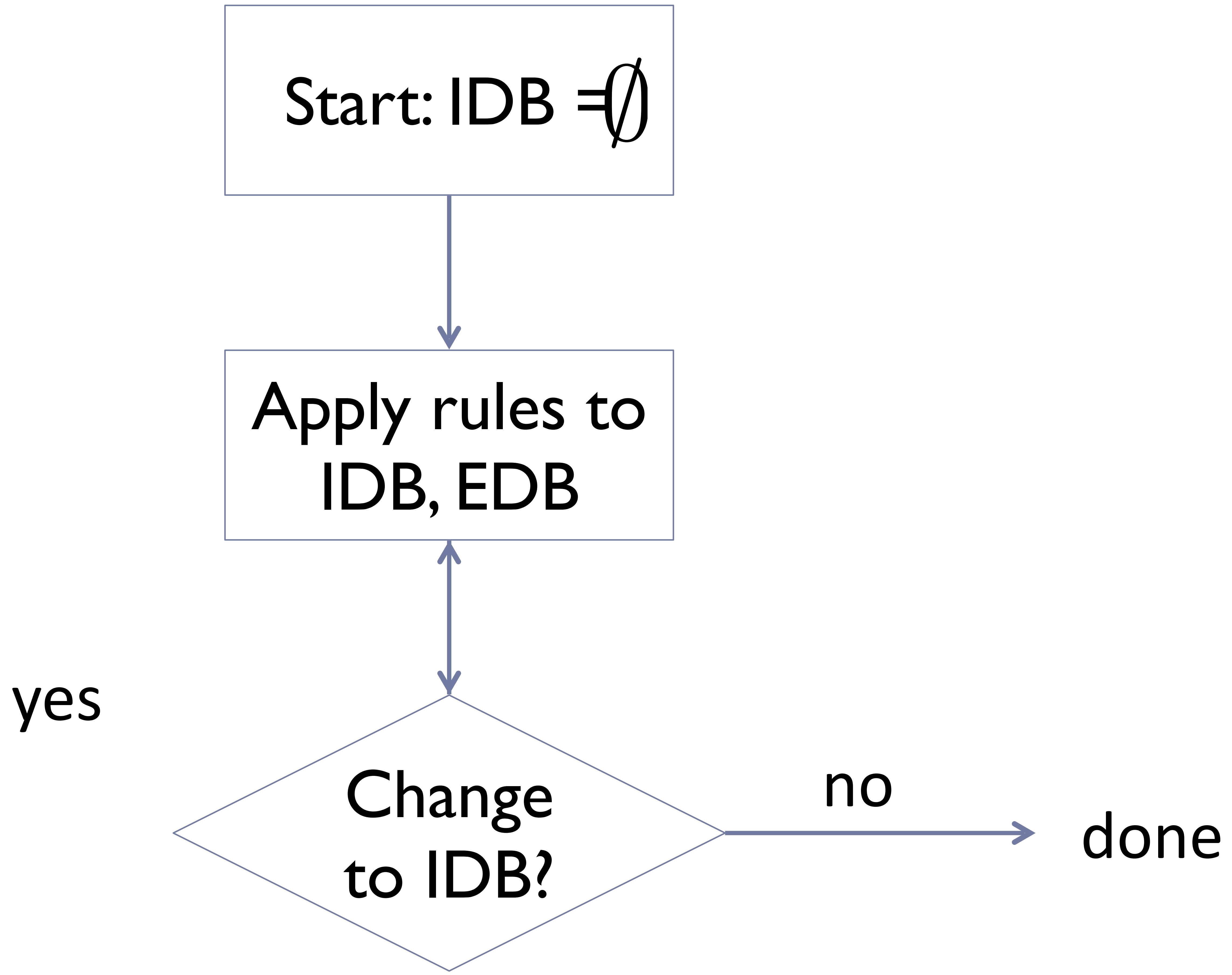
Meaning of datalog rules

- 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



Semi-naïve evaluation

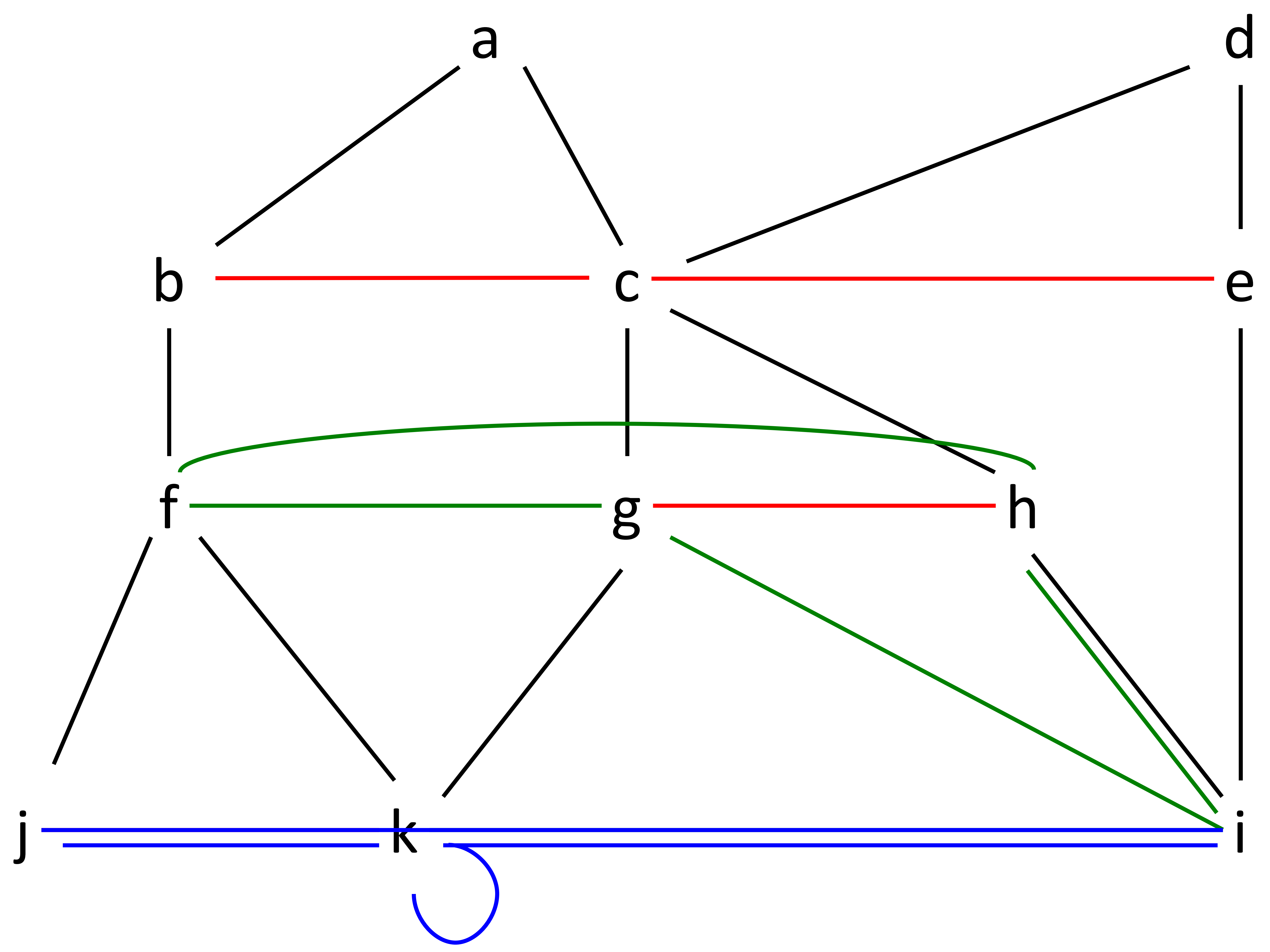
- Since the EDB never changes, on each round we only get new IDB tuples if we use at least one IDB tuple obtained in the previous round
- Saves work; lets us avoid re-discovering most known facts
  - Though a fact can still be derived in more than one way

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

P(x) :- R(x) & ¬Q(x)
Q(x) :- R(x) & ¬P(x)

Example

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

Stratum

- 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→ 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

- Not stratified

P(x) :- R(x) & ¬Q(x)
Q(x) :- R(x) & ¬P(x)

$$\begin{tikzcd} P \arrow[r, bend left, "-"] & Q \arrow[l, bend left, "-"] \end{tikzcd}$$

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