# CSE 544: Principles of Database Systems

## Query Complexity

# Announcements

- Project
  - I started to email feedback, will continue today
  - Milestone due this coming Sunday
  - You are working hard on the project this week!
- Reading assignments
  - None this week
  - Optional reading: two books

# Brief Review of Datalog

- Discuss the naïve and semi-naïve algorithm

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

- Discuss semantics:
  - Minimal model
  - Least fix point
  - Proof theoretic

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

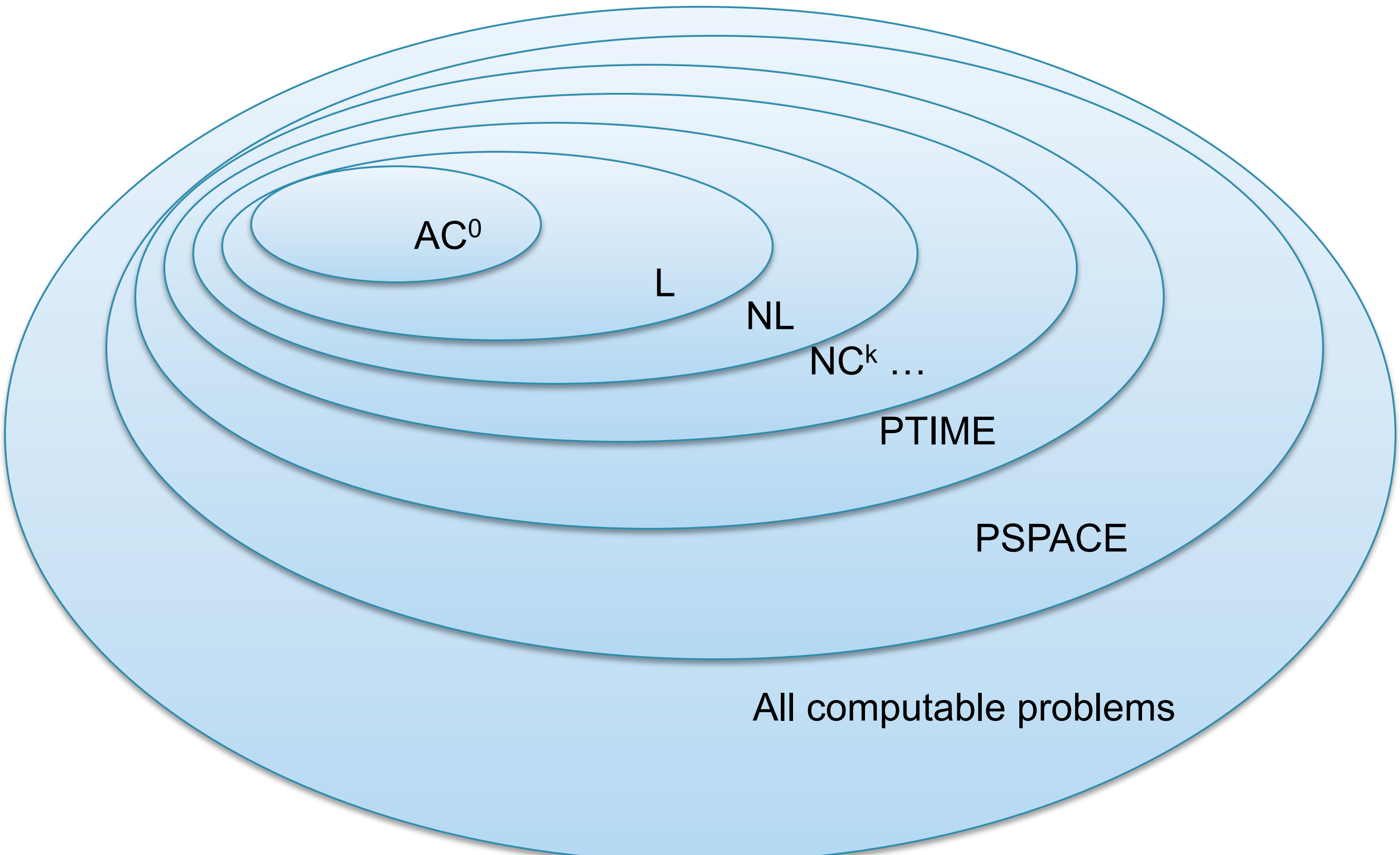
- Adding negation: discuss

# Crash Review of Complexity Classes

In class: define and discuss these complexity classes

- $AC^0$
- L = a.k.a. LOGSPACE
- NL= a.k.a. NLOGSPACE
- NC
- P = a.k.a. PTIME
- NP
- PSPACE

# Crash Review of Complexity Classes



# Crash Review of Complexity Classes

There is one problem that was proven to be outside of $AC^0$

- Which one?

Other classes have not been separated, but have *complete problems*

- What is a complete problem in L?
- What is a complete problem in NL?
- What is a complete problem in NC?
- What is a complete problem in P?
- What is a complete problem in NP?
- What is a complete problem in PSPACE?

# Crash Review of Complexity Classes

There is one problem that was proven to be outside of $AC^0$

- Which one? Parity

Other classes have not been separated, but have *complete problems*

- Complete problem in L: Deterministic reachability. Given a directed, deterministic* graph, two nodes a,b, check if b is reachable from a

- Complete problem in NL: Reachability. Given a directed graph and two nodes a,b, check if b is reachable from a

- Complete problem in NC: N/A (it would be complete for some $NC^k$)

- Complete problems in P: same problem with many names. Alternating Graph Reachability, Circuit Value Problem, Win-Move game, Non-emptyness of a CFG (in class)
- Complete problem in NP: you know these…
- Complete problem in PSPACE: Qantified Boolean Expressions

* A graph is *deterministic* if each node has at most one outgoing edge.

# Crash Review of Complexity Classes

Rules of thumb for dealing with complexity classes

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

Of special interest are problems that are PTIME-complete. Theory tells us that these are *not efficiently parallelizable!*

# The Query Complexity Problem

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

- The answer depends on the query language:
  - Relational calculus, relational algebra
  - Datalog, in various flavors
- Query language design tradeoff
  - High complexity → can express rich queries
  - Low complexity → can be implemented efficiently

# Vardi, *The Complexity of Relational Query Languages*, STOC 1982

Query Q, database D

Moshe Vardi
2008 ACM SIGMOD
Codd Innovation Award

- <u>Data complexity</u>:
  fix Q, complexity = f(D)

- <u>Query complexity</u>:
  fix D, complexity = f(Q)

- <u>Combined complexity</u>:
  complexity = f(D,Q)

# Example

$$Q(x) = \exists y.R(x,y) \land (\forall z.S(y,z) \rightarrow \exists u.R(z,u))$$

Database **D**:

R =

| | |
|---|---|
| a | b |
| a | c |
| b | f |
| b | g |
| a | f |

S =

| | |
|---|---|
| c | b |
| a | c |
| a | a |
| b | a |
| b | b |
| g | b |
| f | a |

Give an algorithm for computing Q on any input **D**.
Express its complexity as a function of* n = |ADom(D)|

* Active Domain = all constants in D. ADom(D) = {a,b,c,…}

# Example

$$Q(x) = \exists y.R(x,y) \wedge (\forall z.S(y,z) \rightarrow \exists u.R(z,u))$$

```
for x in ADom do
   good_y = false
   for y in ADom do
      if (x,y)∈R then
           good_z = true
           for z in ADom do
              if (y,z) ∈ S then
                   good_u = false
                   for u in Adom do
                      if (z,u) ∈ R then good_u = true
                   endfor
                   if not good_u then good_z = false
           endfor
           if good_z then good_y = true
   endfor
   if good_y then output x
endfor
```

Number of variables

Complexity = $O(n^4)$

PTIME

# Conventions

- The complexity is usually defined for a decision problem
  - Hence, we will study only the complexity of Boolean queries
- The complexity usually assumes some encoding of the input
  - Hence we will encode the database instances using a binary represenation

# Boolean Queries

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

Non-boolean queries

$Q(x,y) = \exists z.R(x,z) \wedge S(z,y)$

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

$Q(x)=\exists y.R(x,y) \wedge (\forall z.S(y,z) \rightarrow \exists u.R(z,u))$

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

Boolean queries:

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

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

$Q=\exists y.R('a',y) \wedge (\forall z.S(y,z) \rightarrow \exists u.R(z,u))$

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

# Database Encoding

Encode **D** = (D, $R_1^D$, …, $R_k^D$) as follows:

- Let n = |ADom(D)|
- If $R_i$ has arity k, then encode it as a string of $n^k$ bits:
  - 0 means element $(a_1,\ldots, a_k) \notin R_i^D$
  - 1 means element $(a_1,\ldots, a_k) \in R_i^D$

| | |
|---|---|
| a | b |
| a | c |
| b | b |
| b | c |
| c | a |

| | | |
|---|---|---|
| 0 | 1 | 1 |
| 0 | 1 | 1 |
| 1 | 0 | 0 |

# The Data Complexity

Fix any Boolean query Q in the query language. Determine the complexity of the following problem:

- Given an input database instance **D** = (D, $R_1^D$, …, $R_k^D$), check if Q(**D**) = true.

- This is also known as the *<u>Model Checking Problem</u>*: check if **D** is a model for Q.

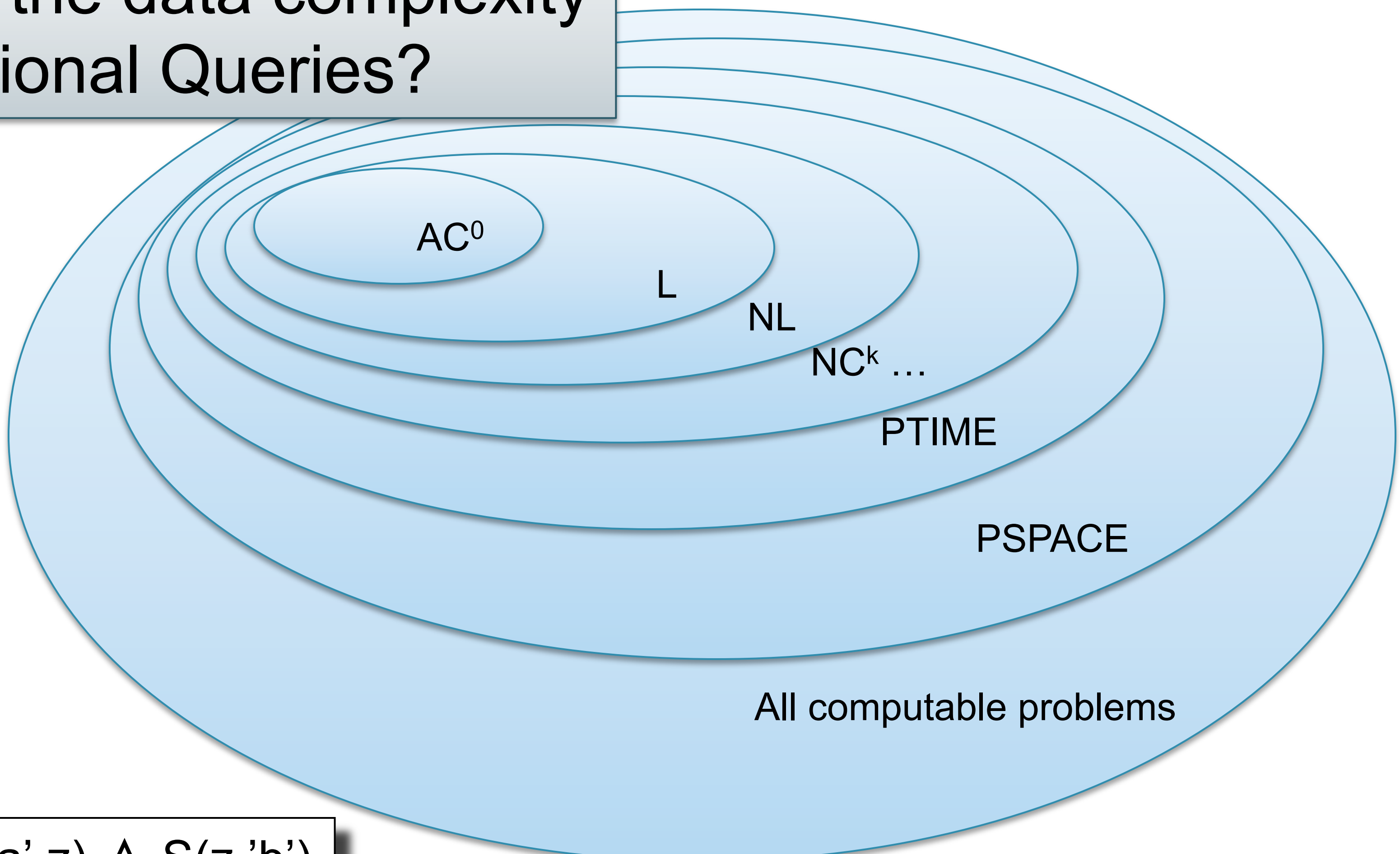
# The Data Complexity

Will discuss next:

- Relational queries
- Datalog and stratified datalog$^{\neg}$
- Datalog$^{\neg}$ with inflationary fixpoint
- Datalog$^{\neg}$ with partial fixpoint

# Question in Class

What is the data complexity of Relational Queries?




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

# Example

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

Prove that Q is in $AC^0$

| R: | a | b | c |
|---|---|---|---|
| a | 0 | 1 | … |
| b | | | |
| c | | | |

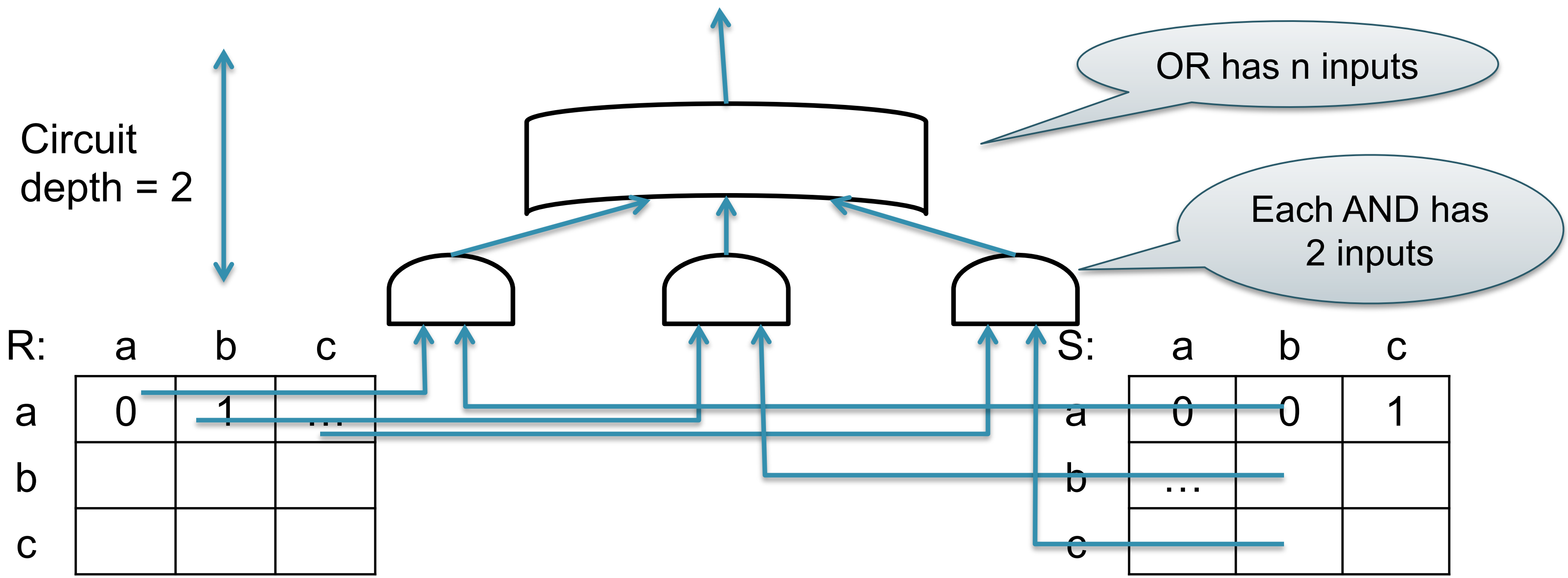
| S: | a | b | c |
|---|---|---|---|
| a | 0 | 0 | 1 |
| b | … | | |
| c | | | |

# Example

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

Prove that Q is in $AC^0$

Example circuit for n = 3 (i.e. ADom={a,b,c})



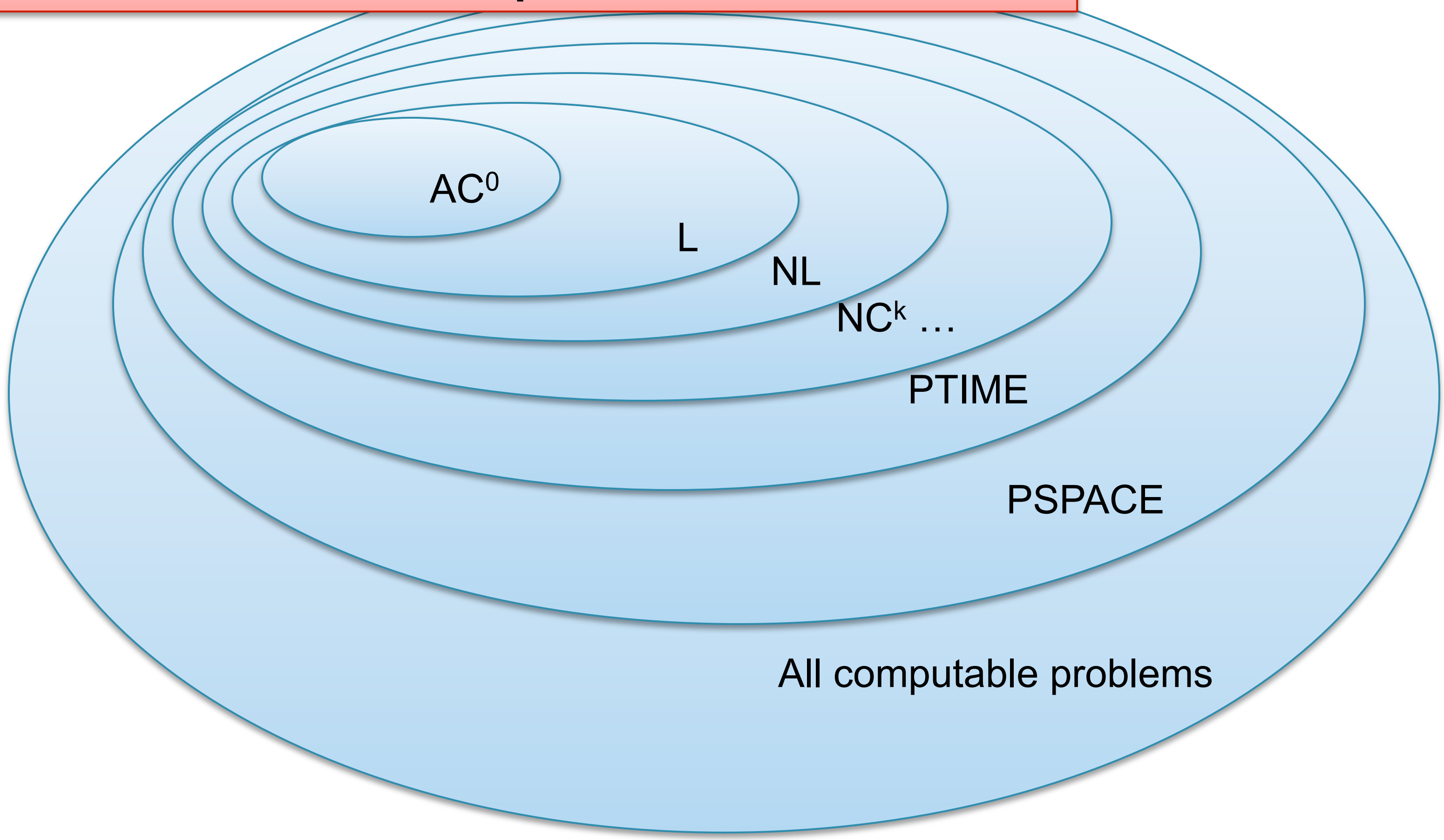
# Another Example

$$Q = \exists y.R(\text{'a'},y) \wedge (\forall z.S(y,z) \rightarrow \exists u.R(z,u))$$

Practice at home:
Show that Q is in $AC^0$ by showing how to construct a circuit for computing Q.
What is the depth?
What fanouts have your OR and AND gates ?

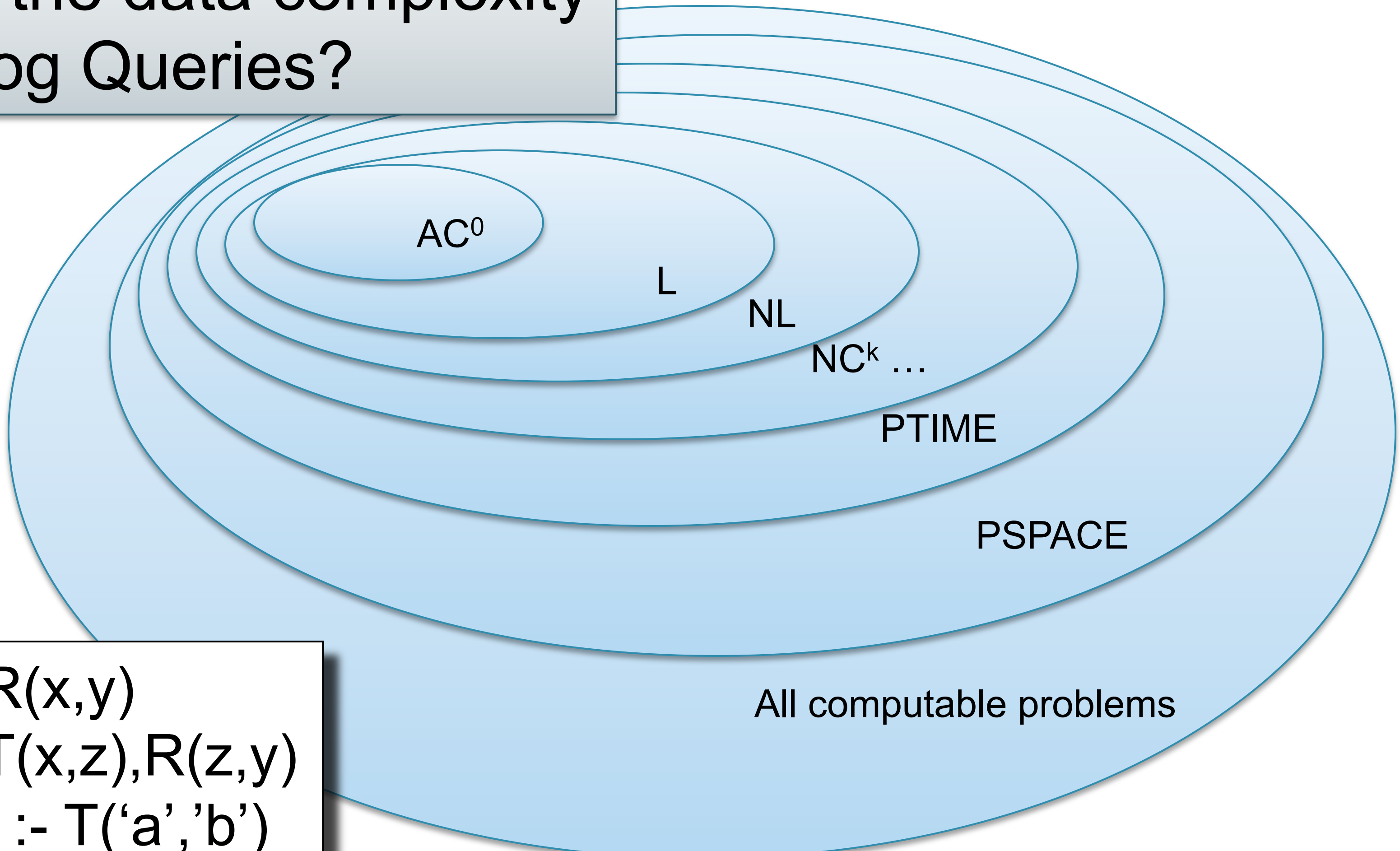
# Relational Queries

**Theorem.** All relational queries are in $AC^0$



# Question in Class

What is the data complexity of datalog Queries?



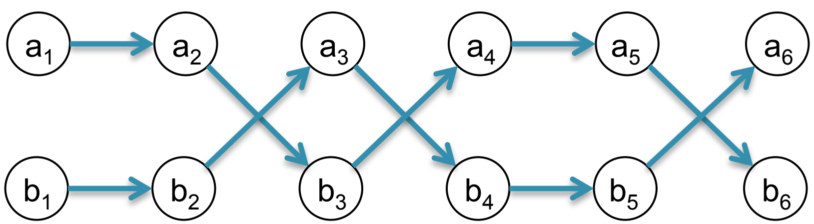

```
T(x,y) :- R(x,y)
T(x,y) :- T(x,z),R(z,y)
Answer() :- T(‘a’,’b’)
```

# Datalog

**Datalog is not in $AC^0$**

Recall that "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 1's is odd
iff Answer is true

# Datalog

**Theorem**. Datalog is in PTIME.

More precisely, fix any Boolean datalog program P.
The problem: given D, check if P(D) = true is in PTIME

Proof: … [discuss in class]

# Datalog

**Theorem**. Datalog is in PTIME.

More precisely, fix any Boolean datalog program P.
The problem: given D, check if P(D) = true is in PTIME

Proof: … [discuss in class]

Each iteration of the naïve algorithm is in PTIME (in fact, in $AC^0$)

```
P_1 = P_2 = … = ∅
Loop
    NewP_1 = SPJU_1; NewP_2 = SPJU_2;  …
    if (NewP_1 = P_1 and NewP_2 = P_2 and …)
        then break
    P_1 = NewP_1; P_2 = NewP_2; …
Endloop
```

If an IDB $P_i$ has arity k, then it will reach its fixpoint after at most $n^k$ iterations. Hence, it is in PTIME.

# Stratified and Inflationary Datalog$^\neg$

**Theorem**. Stratified datalog$^\neg$ is in PTIME.

Why?

**Theorem**. datalog$^\neg$ with inflationary semantics is in PTIME.

Why?

# Datalog

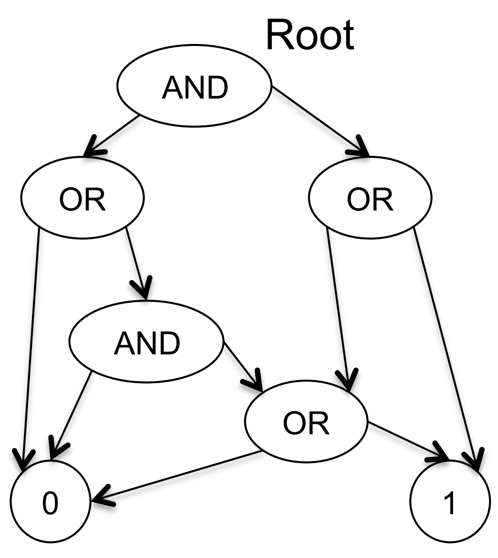
**Datalog can express the Circuit Value Problem**

Circuit value:
Input = a rooted DAG; leaves labeled 0/, internal nodes labeled AND/OR/NOT,
Output = check if the value of the root is 1

Note: assume w.l.o.g. that the circuit is in Negation Normal Form,
i.e. all negations are pushed to the leaves (where 0→1 and 1→0)

## Write datalog program over EDB:

ROOT(x)
AND(x,y1,y2)
OR(x,y1,y2)
ZERO(x)
ONE(x)



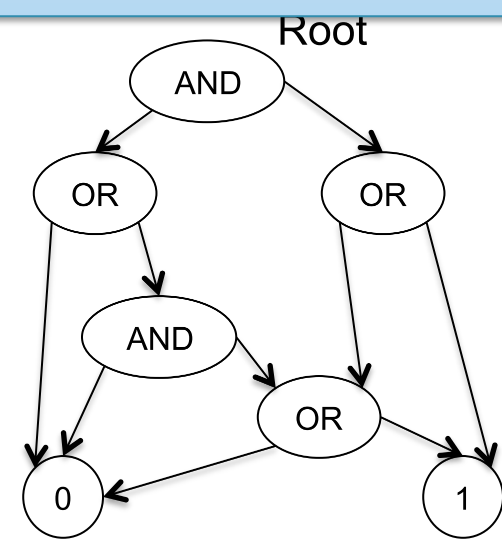
# Datalog

Datalog can expression the Circuit Value Problem

IsOne(x) :- ONE(x)
IsOne(x) :- OR(x,y1,y2),IsOne(y1)
IsOne(x) :- OR(x,y1,y2),IsOne(y2)
IsOne(x) :- AND(x,y1,y2),IsOne(y1),IsOne(y2)
Answer() :- ROOT(x), IsOne(x)

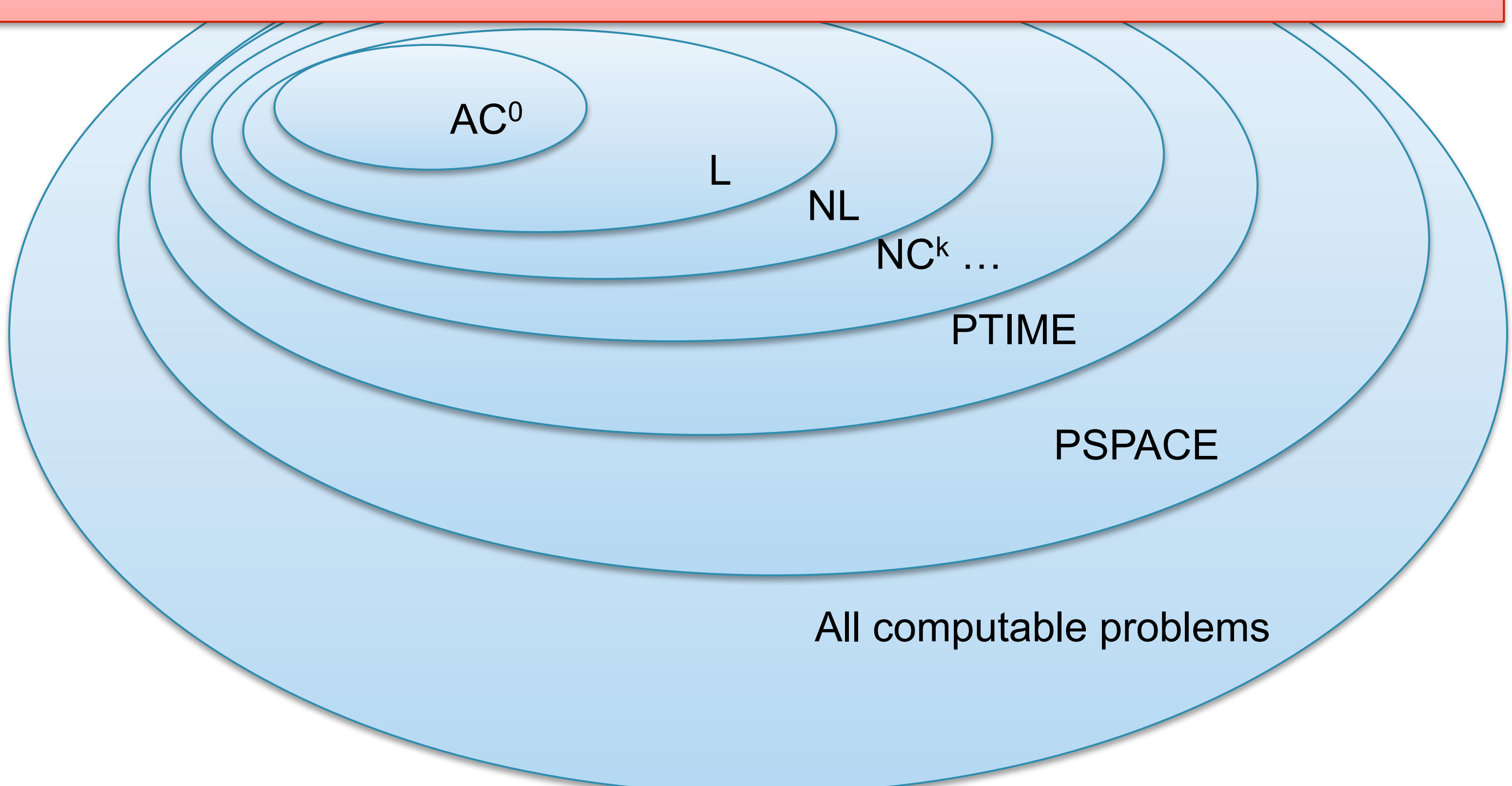
ROOT(x)
AND(x,y1,y2)
OR(x,y1,y2)
ZERO(x)
ONE(x)

Discuss the "move-win" game in class.



# Datalog, stratified and inflationary datalog¬

**Theorem**. Datalog, stratified and inflationary datalog¬ are in PTIME



# NLogspace

Some datalog programs are in NL ( = Nlogspace)

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

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

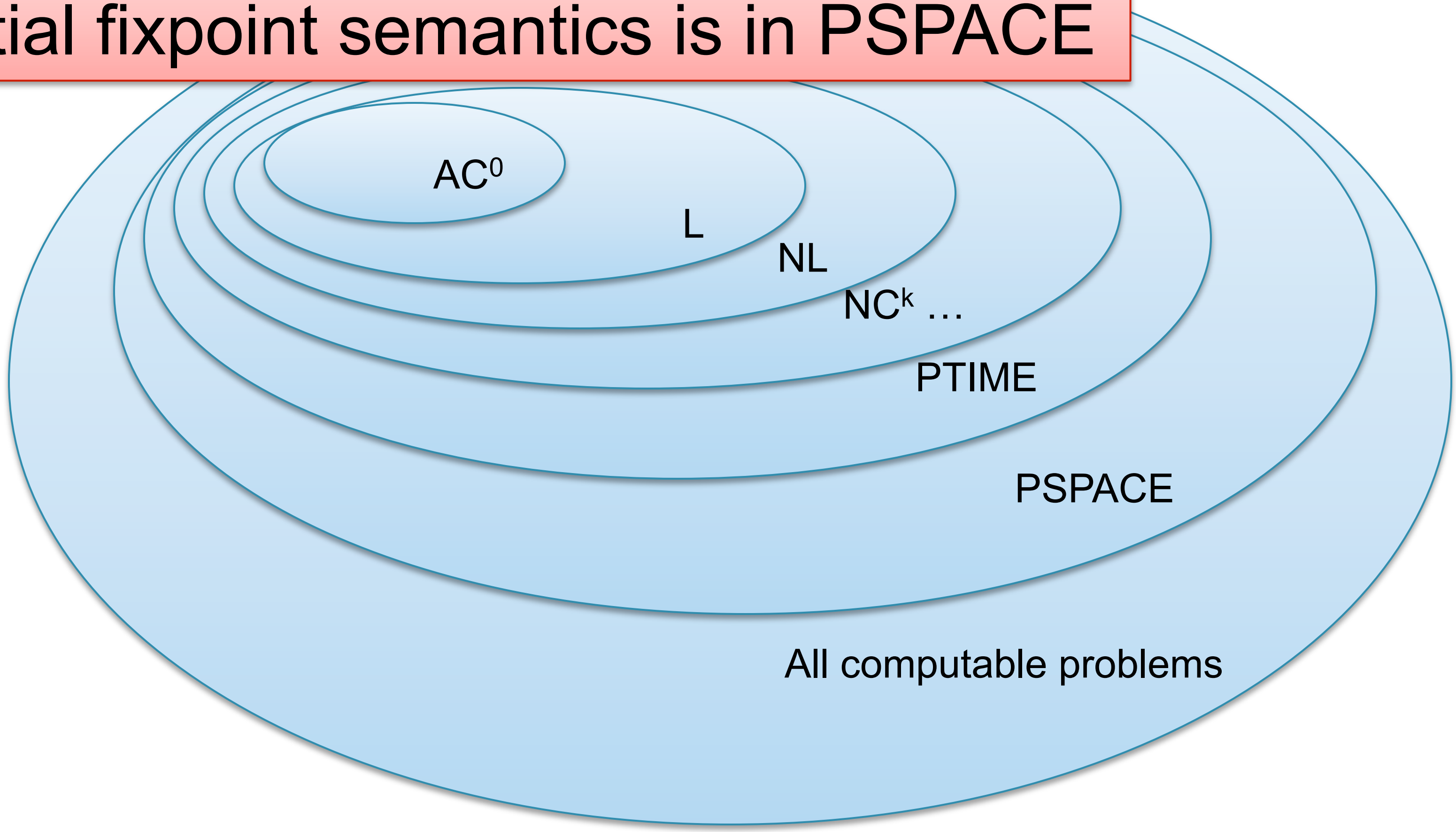
What about the following "same generation" query?
Is it in NL, or is it PTIME complete?

```
S(x,y) :- R(z,x),R(z,y)
S(x,y) :- R(x1,x),R(y1,y),S(x1,y1)
Answer() :- S('a','b')
```

Answer at home…

# Partial Fixpoint Datalog$^\neg$

**Theorem**. Datalog$^\neg$
with partial fixpoint semantics is in PSPACE



# Descriptive Complexity

- In computational complexity one describes complexity classes in terms of a computational model
  - Turing Machine, circuit, etc
- In descriptive complexity one describes complexity classes in terms of the logic ("query language") that captures that class

# Descriptive Complexity

Assume we have access to an order relation <
(and to a BIT relation for $AC^0$)