CSE 544: Principles of Database Systems

Semijoin Reductions

Theory Wrap-up

Announcements

- Makeup lectures:
  - Friday, May 18, 10:30-11:50, CSE 405
  - Friday, May 25, 10:30-11:50, CSE 405
- No lectures:
  - Monday and Wednesday (May 21 and 23)
- Updated time:
  - Wednesday, May 30, 9-10:30, CSE 405
- Paper reviews
  - May 25: provenance
  - May 30: privacy
- Project presentations:
  - Monday, May 28, 1:30-4:30 and
  - Tuesday, May 29, 8:30-12
- Homework 3:
  - Coming today…
  - Due Sunday, June 3 at midnight

Outline

- Semijoin reductions
- Theory wrapup

Law of Semijoins

Recall the definition of a semijoin:

- $R \ltimes S = \Pi_{A1,\ldots,An} (R \bowtie S)$
- Where the schemas are:
  - Input: R(A1,…An), S(B1,…,Bm)
  - Output: T(A1,…,An)
- The law of semijoins is:

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

Remark

- Prove that the following two non-recursive datalog queries are equivalent:

$Q_1(x,y,z) = R(x,y),S(x,z)$

$R_1(x,y) = R(x,y),S(x,z)$
$Q_2(x,y,z) = R_1(x,y),S(x,z)$

Remark

- Prove that the following two non-recursive datalog queries are equivalent:

$$Q_1(x,y,z) = R(x,y),S(x,z)$$

$$R_1(x,y) = R(x,y),S(x,z)$$
$$Q_2(x,y,z) = R_1(x,y),S(x,z)$$

$$Q_1(x,y,z) = R(x,y),S(x,z)$$

$$Q_2(x,y,z) = R(x,y),S(x,u),S(x,z)$$

$$Q_1(x,y,z) = R(x,y),S(x,z)$$

$$Q_2(x,y,z) = R(x,y),S(x,u),S(x,z)$$

Laws with Semijoins

- Very important in parallel databases
- Often combined with Bloom Filters
- See pp. 747 in the textbook

Semijoin Reducer

- Given a query:

$$Q = R_1 \bowtie R_2 \bowtie \ldots \bowtie R_n$$

- A *semijoin reducer* for Q is

$$R_{i1} = R_{i1} \ltimes R_{j1}$$
$$R_{i2} = R_{i2} \ltimes R_{j2}$$
$$\ldots\ldots$$
$$R_{ip} = R_{ip} \ltimes R_{jp}$$

  such that the query is equivalent to:

$$Q = R_{k1} \bowtie R_{k2} \bowtie \ldots \bowtie R_{kn}$$

- A *full reducer* is such that no dangling tuples remain

Example

- Example:

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

- A semijoin reducer is:

$$R_1(A,B) = R(A,B) \ltimes S(B,C)$$

- The rewritten query is:

$$Q = R_1(A,B) \bowtie S(B,C)$$

Why Would We Do This ?

- Reduce amount of communication

$$Q = \gamma_{A,B,count(*)} R(A,B,D) \ltimes_B \sigma_{C=value}(S(B,C))$$

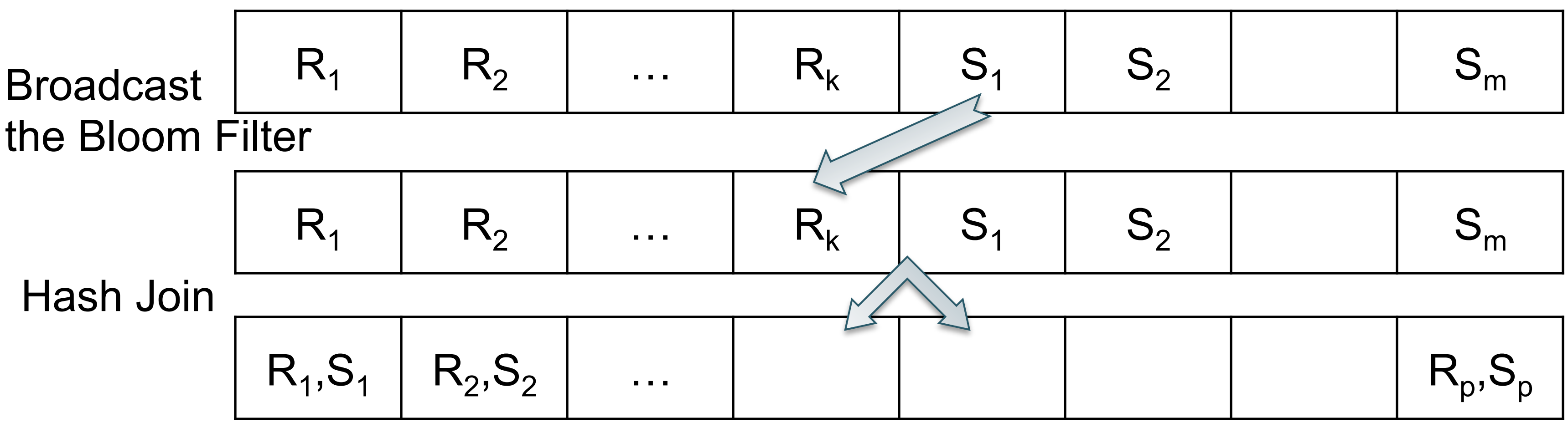
R_1	R_2	…	R_k	S_1	S_2		S_m

How can we optimize this query in a distributed computation?

Why Would We Do This ?

- Reduce amount of communication

$$Q = \gamma_{A,B,count(*)} R(A,B,D) \bowtie_B \sigma_{C=value}(S(B,C))$$



$$R_1(A,B,D) = R(A,B,D) \ltimes_B \sigma_{C=value}(S(B,C))$$
$$Q = \gamma_{A,B,count(*)} R_1(A,B,D) \bowtie_B \sigma_{C=value}(S(B,C))$$

Semijoin Reducer

- Example:

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

- A semijoin reducer is:

$$R_1(A,B) = R(A,B) \ltimes S(B,C)$$

- The rewritten query is:

$$Q = R_1(A,B) \bowtie S(B,C)$$

Are there dangling tuples ?

Semijoin Reducer

- Example:

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

- A full semijoin reducer is:

$$R_1(A,B) \;=\; R(A,B) \ltimes S(B,C)$$
$$S_1(B,C) \;=\; S(B,C) \ltimes R_1(A,B)$$

- The rewritten query is:

$$Q \text{ :- } R_1(A,B) \bowtie S_1(B,C)$$

No more dangling tuples

Semijoin Reducer

- More complex example:

$$Q = R(A,B) \bowtie S(B,C) \bowtie T(C,D,E)$$

- What is a full reducer?

Semijoin Reducer

- More complex example:

$$Q = R(A,B) \bowtie S(B,C) \bowtie T(C,D,E)$$

- A full reducer is:

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

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

Semijoin Reducer

- Example:

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

- Doesn't have a full reducer (we can reduce forever)

Theorem a query has a full reducer iff it is "acyclic" [*Database Theory*, by Abiteboul, Hull, Vianu]

Example with Semijoins

Emp(eid, ename, sal, did)
Dept(did, dname, budget)
DeptAvgSal(did, avgsal) /* view */

[Chaudhuri'98]

View:

```
CREATE VIEW DepAvgSal As (
        SELECT E.did, Avg(E.Sal) AS avgsal
        FROM Emp E
        GROUP BY E.did)
```

Query:

```
SELECT E.eid, E.sal
FROM Emp E, Dept D, DepAvgSal V
WHERE E.did = D.did AND E.did = V.did
        AND E.age < 30 AND D.budget > 100k
        AND E.sal > V.avgsal
```

Goal: compute only the necessary part of the view

Example with Semijoins

[Chaudhuri'98]

Emp(eid, ename, sal, did)
Dept(did, dname, budget)
DeptAvgSal(did, avgsal) /* view */

New view uses a reducer:

```
CREATE VIEW LimitedAvgSal As (
        SELECT E.did, Avg(E.Sal) AS avgsal
        FROM Emp E, Dept D
        WHERE E.did = D.did AND D.buget > 100k
        GROUP BY E.did)
```

New query:

```
SELECT E.eid, E.sal
FROM Emp E, Dept D, LimitedAvgSal V
WHERE E.did = D.did AND E.did = V.did
        AND E.age < 30 AND D.budget > 100k
        AND E.sal > V.avgsal
```

Example with Semijoins

Emp(eid, ename, sal, did)
Dept(did, dname, budget)
DeptAvgSal(did, avgsal) /* view */

[Chaudhuri'98]

Full reducer:

```
CREATE VIEW PartialResult AS
        (SELECT E.eid, E.sal, E.did
        FROM Emp E, Dept D
        WHERE E.did=D.did AND E.age < 30
        AND D.budget > 100k)

CREATE VIEW Filter AS
        (SELECT DISTINCT P.did FROM PartialResult P)

CREATE VIEW LimitedAvgSal AS
        (SELECT E.did, Avg(E.Sal) AS avgsal
        FROM Emp E, Filter F
        WHERE E.did = F.did GROUP BY E.did)
```

Example with Semijoins

New query:

```
SELECT P.eid, P.sal
FROM PartialResult P, LimitedDepAvgSal V
WHERE P.did = V.did AND P.sal > V.avgsal
```

Theory Wrap-up

- Datalog
- Datalog$^\neg$
- Query complexity
- Query containment/equivalence
- Static optimizations (semijoin reductions)

Datalog

What is the motivation behind datalog?

Datalog

What is the motivation behind datalog?

- SQL is declarative (great) but limited:
  - Can’t express transitive closure
- Need to extend the declarative paradigm beyond traditional database computations
  - Massive distributed computations
  - Programming on multicores
- Datalog adds recursion to declarative programming

Datalog Key Concepts

- What are the three equivalent semantics in datalog?

- What are the “standard” evaluation algorithms for datalog?
  –

Datalog Key Concepts

- What are the three equivalent semantics in datalog?
  - Minimal model semantics
  - Least fixpoint semantics
  - Proof-theoretic semantics
- What are the "standard" evaluation algorithms for datalog?
  - Naïve algorithm
  - Semi-naïve algorithm

Datalog¬

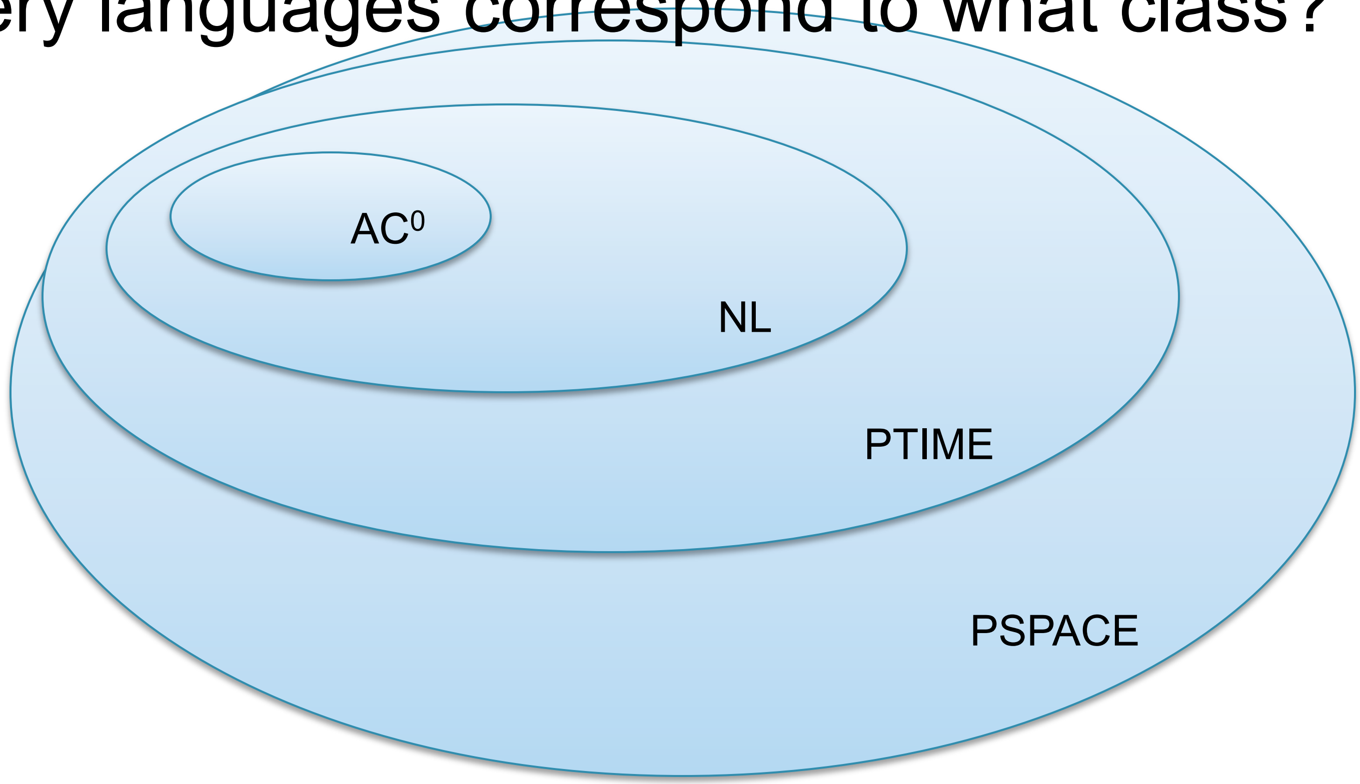
- Recursion and negation don’t mix!
  - Why?
- What are the three different semantics of Datalog¬?
-

Datalog$^{\neg}$

- Recursion and negation don’t mix!
  - Why?
- What are the three different semantics of Datalog$^{\neg}$?
  - Stratified Datalog$^{\neg}$
  - Inflationary fixpoint
  - Partial fixpoint
- Increasing expressive power (see HW3)

Query Complexity

What are these complexity classes?
What do they mean from a practical point of view?
Which query languages correspond to what class?

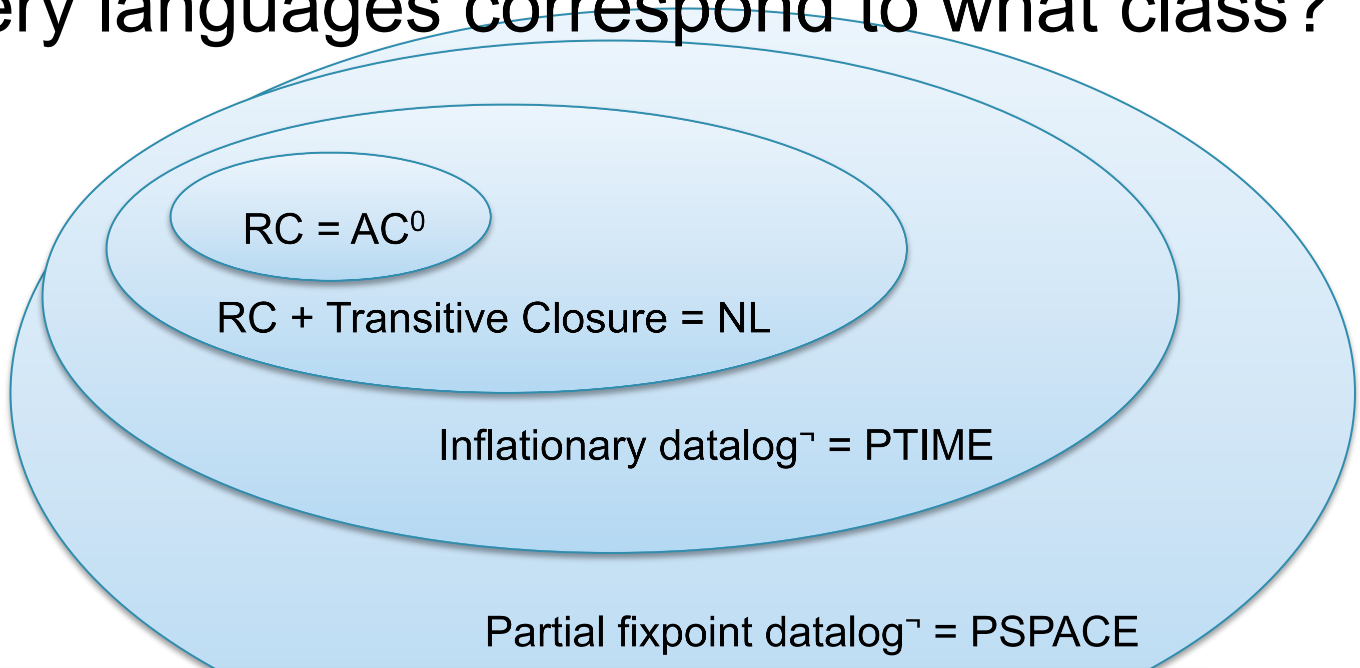


Query Complexity

What are these complexity classes?

What do they mean from a practical point of view?

Which query languages correspond to what class?



AC^0 = embarrassingly parallel
NL = some iteration required, theoretically parallel
PTIME = efficient, no longer parallel
PSPACE = potentially inefficient, needs careful programming (Dedalus)

Query Containment/Equivalence

- Can we check whether two Java functions compute the same (mathematical) function?

- Can we check whether two conjunctive queries compute the same (mathematical) function?
  –

Query Containment/Equivalence

- Can we check whether two Java functions compute the same (mathematical) function?
  - No: it is undeciable (by Rice's theorem)

- Can we check whether two conjunctive queries compute the same (mathematical) function?
  - Yes: the problem is NP-complete

Query Containment/Equivalence

- What are the two equivalent criteria for checking query containment $q_1 \subseteq q_2$?

Query Containment/Equivalence

- What are the two equivalent criteria for checking query containment $q_1 \subseteq q_2$?
  - Check if q_2 returns the canonical tuple on the canonical database for q_1
  - Check if there exists a homomorphism $q_2 \rightarrow q_1$

Query Containment/Equivalence

- CQ – NP-complete
- Unions of CQ – NP-complete
- $CQ^<$ – Π^p_2 complete
- Relational Calculus – undecidable

- Trakhtentbrot's theorem = implies that there is virtually nothing one can decide about the semantics of RC queries

Static Optimizations

- Semijoin reductions
  - Very important in Big Data processing
  - Often combined with Bloom filters (what are they?)
- Magic sets
  - These are semijoin reductions for datalog programs
  - In HW3 you will be asked to do manually a semijoin reduction