CSEP 544

Lecture 8: Datalog

Announcements

- Homework 4 was due last night
  - We need to delete your database after you submit
- Homework 5 is posted
  - Due on Monday, March 10
  - Last homework!
- Reading assignment due today:
  - Datalog
- Reading assignment due on March 11:
  - Column-oriented databases (long!)

Outline for Today

- Datalog
- Provenance (will continue next time)

Paper Discussion

- Why the *urgency of parallelism?*

Paper Discussion

- Why the *urgency of parallelism?*
- Moore's Law: transistor density continues to grow exponentially, but no longer improving processor speeds. Instead: multicores
- Cloud Computing: commoditize access to large compute clusters:

Datalog

Review (from Lecture 2)

- Fact
- Rule
- Head and body of a rule
- Existential variable
- Head variable

Review

Facts

Rules

```
Actor(344759,‘Douglas’, ‘Fowley’).
Casts(344759, 29851).
Casts(355713, 29000).
Movie(7909, ‘A Night in Armour’, 1910).
Movie(29000, ‘Arizona’, 1940).
Movie(29445, ‘Ave Maria’, 1940).
```

Facts = tuples in the database
Rules = queries

Review

Facts

Actor(344759,‘Douglas’, ‘Fowley’).
Casts(344759, 29851).
Casts(355713, 29000).
Movie(7909, ‘A Night in Armour’, 1910).
Movie(29000, ‘Arizona’, 1940).
Movie(29445, ‘Ave Maria’, 1940).

Rules

Q1(y) :- Movie(x,y,z), z=‘1940’.

Facts = tuples in the database
Rules = queries

Review

Facts

Actor(344759,‘Douglas’, ‘Fowley’).
Casts(344759, 29851).
Casts(355713, 29000).
Movie(7909, ‘A Night in Armour’, 1910).
Movie(29000, ‘Arizona’, 1940).
Movie(29445, ‘Ave Maria’, 1940).

Rules

Q1(y) :- Movie(x,y,z), z=‘1940’.

Q2(f, l) :- Actor(z,f,l), Casts(z,x),
Movie(x,y,’1940’).

Facts = tuples in the database
Rules = queries

Review

Facts

Actor(344759,‘Douglas’, ‘Fowley’).
Casts(344759, 29851).
Casts(355713, 29000).
Movie(7909, ‘A Night in Armour’, 1910).
Movie(29000, ‘Arizona’, 1940).
Movie(29445, ‘Ave Maria’, 1940).

Rules

Q1(y) :- Movie(x,y,z), z=‘1940’.

Q2(f, l) :- Actor(z,f,l), Casts(z,x),
Movie(x,y,’1940’).

Q3(f,l) :- Actor(z,f,l), Casts(z,x1), Movie(x1,y1,1910),
Casts(z,x2), Movie(x2,y2,1940)

Facts = tuples in the database
Rules = queries

Review

Facts

Actor(344759,'Douglas', 'Fowley').
Casts(344759, 29851).
Casts(355713, 29000).
Movie(7909, 'A Night in Armour', 1910).
Movie(29000, 'Arizona', 1940).
Movie(29445, 'Ave Maria', 1940).

Rules

Q1(y) :- Movie(x,y,z), z='1940'.

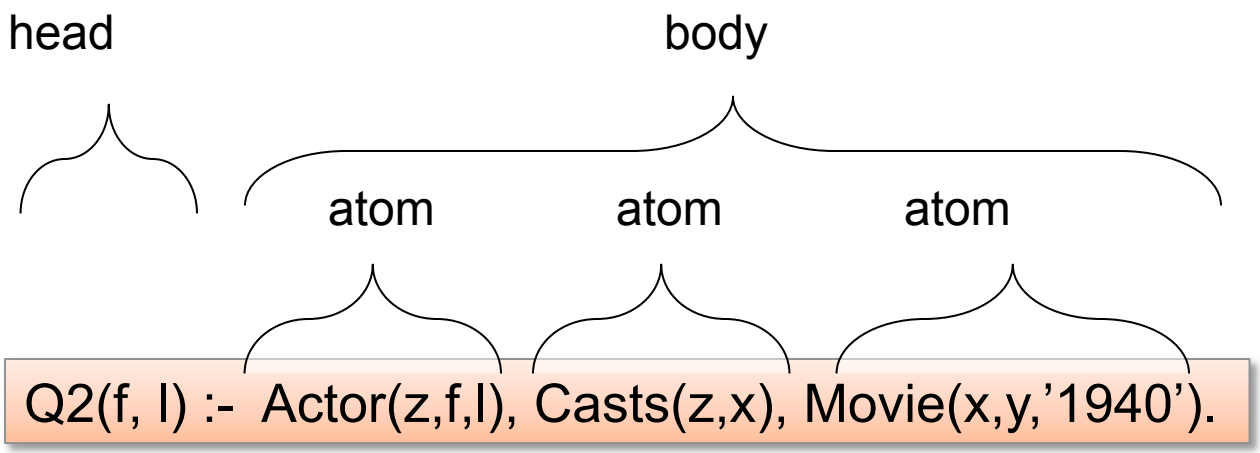
Q2(f, l) :- Actor(z,f,l), Casts(z,x), Movie(x,y,'1940').

Q3(f,l) :- Actor(z,f,l), Casts(z,x1), Movie(x1,y1,1910), Casts(z,x2), Movie(x2,y2,1940)

Facts = tuples in the database
Rules = queries

Extensional Database Predicates = EDB
Intensional Database Predicates = IDB

Review



f, l = head variables
x,y,z= existential variables

Paper Discussion

- How does the Dedalus data model extend the basic datalog model?

- How does Dedalus extend the datalog rules?

Paper Discussion

- How does the Dedalus data model extend the basic datalog model?
  - Each predicate has a timestamp and a location
- How does Dedalus extend the datalog rules?
  - Deductive, inductive, asynchronous rules

Paper Discussion

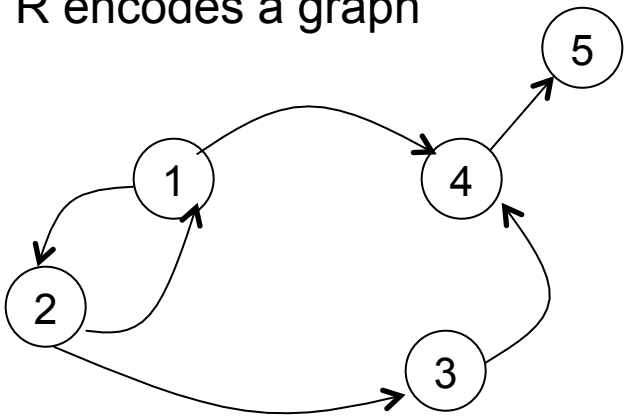
```
toggle(1) :- state(0).                  toggle(1, T) :- state(0, T).
toggle(0) :- state(1).                  toggle(0, T) :- state(1, T).
state(X)@next :- toggle(X).             state(X, S) :- toggle(X, T), succ(T, S).
announce(X)@async :- toggle(X).         announce(X, S) :- toggle(X, T), choice({X,T}, {S}).
```

Figure 1: A simple Dedalus program, written with syntactic sugar (left), and with standard Datalog notation (right).

Which are the deductive, inductive, asynchronous rules?

Simple datalog programs

R encodes a graph




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

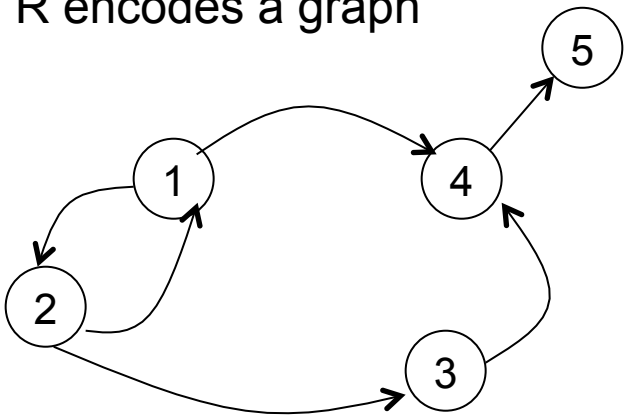
What does it compute?

R=

1	2
2	1
2	3
1	4
3	4
4	5

Simple datalog programs

R encodes a graph



R=

1	2
2	1
2	3
1	4
3	4
4	5

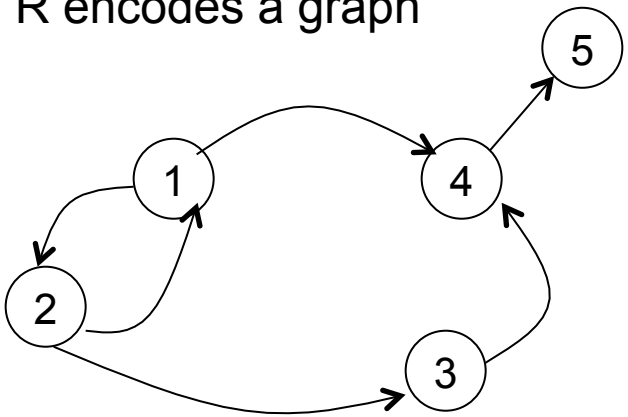
Initially:
T is empty.

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

What does it compute?

Simple datalog programs

R encodes a graph




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

What does it compute?

R=

1	2
2	1
2	3
1	4
3	4
4	5

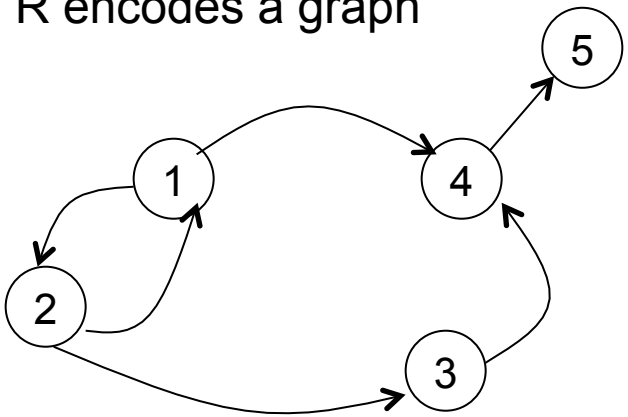
Initially:
T is empty.

First iteration:
T =

1	2
2	1
2	3
1	4
3	4
4	5

Simple datalog programs

R encodes a graph




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

What does it compute?

R=

1	2
2	1
2	3
1	4
3	4
4	5

Initially:
T is empty.

First iteration:
T =

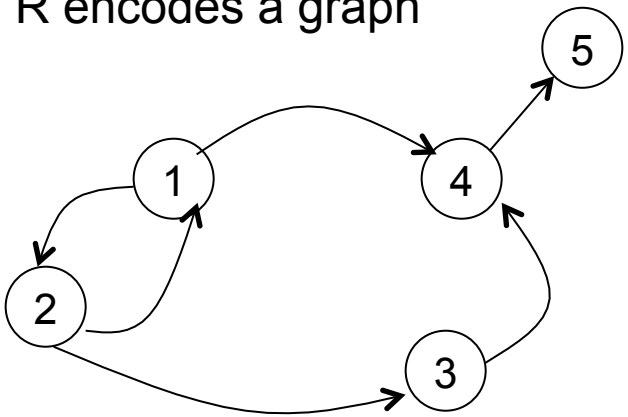
1	2
2	1
2	3
1	4
3	4
4	5

Second iteration:
T =

1	2
2	1
2	3
1	4
3	4
4	5
1	1
2	2
1	3
2	4
1	5
3	5

Simple datalog programs

R encodes a graph




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

What does it compute?

R=

1	2
2	1
2	3
1	4
3	4
4	5

Initially:
T is empty.

First iteration:
T =

1	2
2	1
2	3
1	4
3	4
4	5

Second iteration:
T =

1	2
2	1
2	3
1	4
3	4
4	5
1	1
2	2
1	3
2	4
1	5
3	5

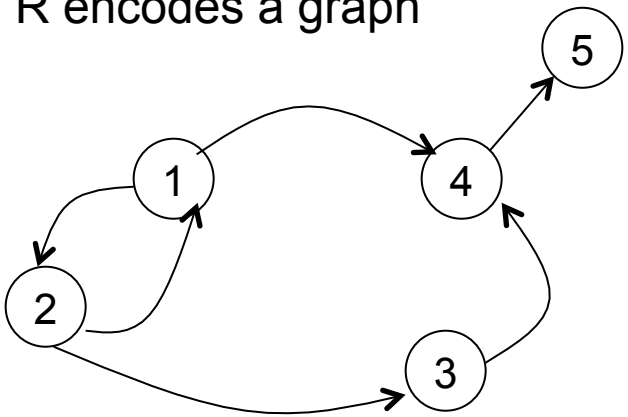
Third iteration:
T =

1	2
2	1
2	3
1	4
3	4
4	5
1	1
2	2
1	3
2	4
1	5
3	5
2	5

Done

Simple datalog programs

R encodes a graph




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

What does it compute?

R=

1	2
2	1
2	3
1	4
3	4
4	5

Initially:
T is empty.

First iteration:
T =

1	2
2	1
2	3
1	4
3	4
4	5

Second iteration:
T =

1	2
2	1
2	3
1	4
3	4
4	5
1	1
2	2
1	3
2	4
1	5
3	5

Third iteration:
T =

1	2
2	1
2	3
1	4
3	4
4	5
1	1
2	2
1	3
2	4
1	5
3	5
2	5

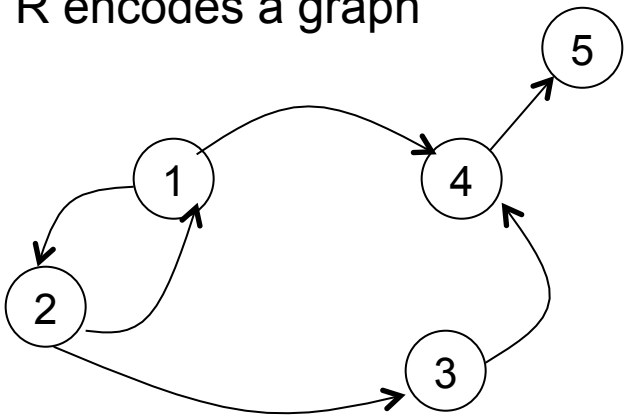
Discovered 3 times!

Discovered twice

Done

Simple datalog programs

R encodes a graph



R=

1	2
2	1
2	3
1	4
3	4
4	5

Alternative ways to compute TC:

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

Right linear

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

Left linear

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

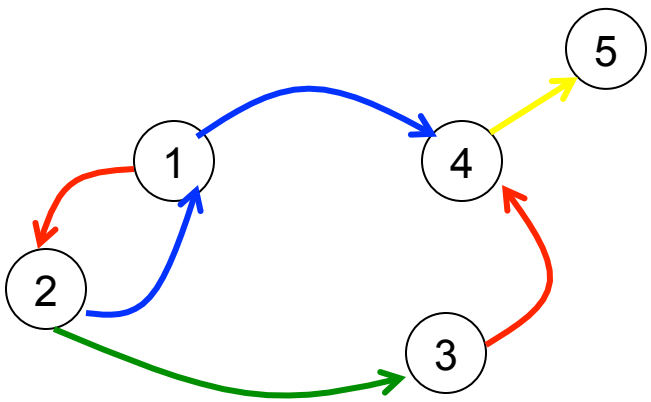
Non-linear

Discuss pros/cons in class

Simple datalog programs

Compute TC (ignoring color):

R encodes a colored graph



Compute pairs of nodes connected by the same color (e.g. (2,4))

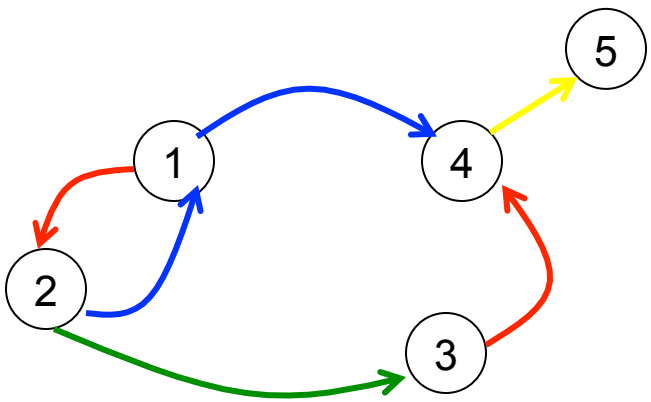
R=

1	Red	2
2	Blue	1
2	Green	3
1	Blue	4
3	Red	4
4	Yellow	5

Simple datalog programs

Compute TC (ignoring color):

R encodes a colored graph




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

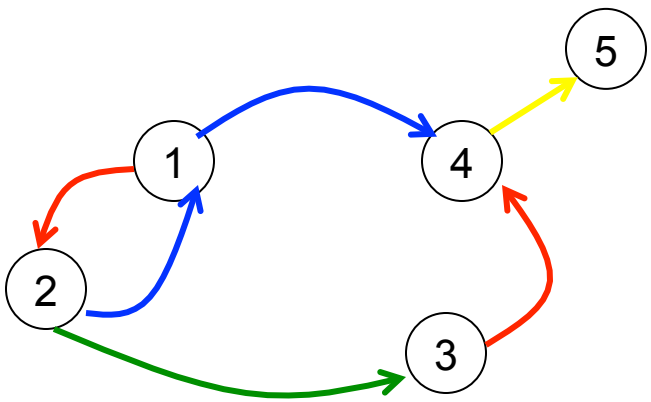
Compute pairs of nodes connected by the same color (e.g. (2,4))

R=

1	Red	2
2	Blue	1
2	Green	3
1	Blue	4
3	Red	4
4	Yellow	5

Simple datalog programs

R encodes a colored graph



R=

1	Red	2
2	Blue	1
2	Green	3
1	Blue	4
3	Red	4
4	Yellow	5

Compute TC (ignoring color):

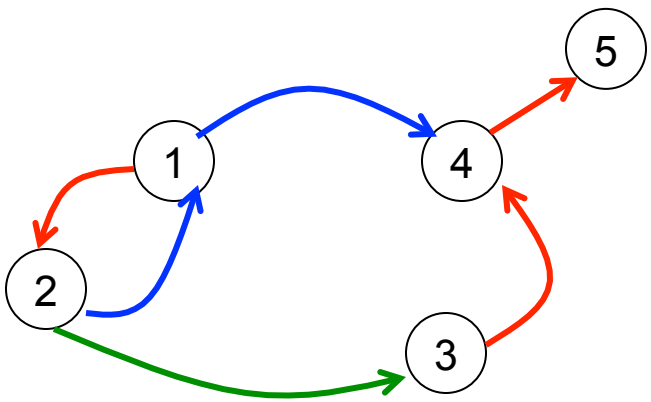
```
T(x,y) :- R(x,c,y)
T(x,y) :- R(x,c,z), T(z,y)
```

Compute pairs of nodes connected by the same color (e.g. (2,4))

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

Simple datalog programs

R, G, B encodes a 3-colored graph



R=

1	2
3	4
4	5

G=

2	3

B=

2	1
1	4

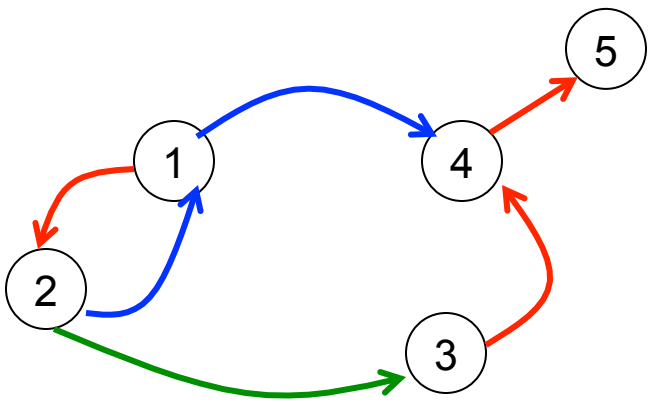
What does this program compute in general?

```
S(x,y) :- B(x,y)
S(x,y) :- T(x,z),B(z,y)
T(x,y) :- S(x,z),R(z,y)
T(x,y) :- S(x,z),G(z,y)
Answer(x,y) :- T(x,y)
```

Simple datalog programs

R, G, B encodes a 3-colored graph

What does this program compute in general?



R=

1	2
3	4
4	5

G=

2	3

B=

2	1
1	4

```
S(x,y) :- B(x,y)
S(x,y) :- T(x,z),B(z,y)
T(x,y) :- S(x,z),R(z,y)
T(x,y) :- S(x,z),G(z,y)
Answer(x,y) :- T(x,y)
```

Answer: it computes pairs of nodes connected by a path spelling out these regular expressions:

- S = (B.(R or G))*.B
- T = (B.(R or G))$^+$

Syntax of Datalog Programs

The schema consists of two sets of relations:

- Extensional Database (EDB): R_1, R_2, …
- Intentional Database (IDB): P_1, P_2, …

A datalog program **P** has the form:

P:

$P_{i1}(x_{11},x_{12},\ldots)$:- $body_1$
$P_{i2}(x_{21},x_{22},\ldots)$:- $body_2$

….

- Each head predicate P_i is an IDB
- Each body is a conjunction of IDB and/or EDB predicates
- See lecture 2

Note: no negation (yet)! Recursion OK.

Naïve Datalog Evaluation Algorithm

Datalog program:

P_{i1} :- $body_1$
P_{i2} :- $body_2$
….

Naïve Datalog Evaluation Algorithm

Datalog program:

P_{i1} :- $body_1$
P_{i2} :- $body_2$
….

➔

Group by
IDB predicate

P_1 :- $body_{11} \cup body_{12} \cup \ldots$
P_2 :- $body_{21} \cup body_{22} \cup \ldots$
….

Naïve Datalog Evaluation Algorithm

Datalog program:

P_{i1} :- $body_1$
P_{i2} :- $body_2$
….

➔

Group by
IDB predicate

P_1 :- $body_{11} \cup body_{12} \cup \ldots$
P_2 :- $body_{21} \cup body_{22} \cup \ldots$
….

➔

Each rule is a
Select-Project-Join-Union query

P_1 :- $SPJU_1$
P_2 :- $SPJU_2$
….

Naïve Datalog Evaluation Algorithm

Datalog program:

P_{i1} :- $body_1$
P_{i2} :- $body_2$
….

➔

Group by
IDB predicate

P_1 :- $body_{11} \cup body_{12} \cup \ldots$
P_2 :- $body_{21} \cup body_{22} \cup \ldots$
….

➔

Each rule is a
Select-Project-Join-Union query

P_1 :- $SPJU_1$
P_2 :- $SPJU_2$
….

Example:

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

➔ ?

Naïve Datalog Evaluation Algorithm

Datalog program:

P_{i1} :- $body_1$
P_{i2} :- $body_2$
….

➔ Group by IDB predicate

P_1 :- $body_{11} \cup body_{12} \cup \ldots$
P_2 :- $body_{21} \cup body_{22} \cup \ldots$
….

➔ Each rule is a Select-Project-Join-Union query

P_1 :- $SPJU_1$
P_2 :- $SPJU_2$
….

Example:

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

➔ $T(x,y) :- R(x,y) \cup \Pi_{xy}(R(x,z) \bowtie T(z,y))$

Naïve Datalog Evaluation Algorithm

Datalog program:

P_{i1} :- $body_1$
P_{i2} :- $body_2$
….

➔

Group by IDB predicate

P_1 :- $body_{11} \cup body_{12} \cup \ldots$
P_2 :- $body_{21} \cup body_{22} \cup \ldots$
….

➔

Each rule is a Select-Project-Join-Union query

P_1 :- $SPJU_1$
P_2 :- $SPJU_2$
….

Naïve datalog evaluation algorithm:

$P_1 = P_2 = \ldots = \varnothing$
Loop
  $NewP_1 = SPJU_1$; $NewP_2 = SPJU_2$; …
  if ($NewP_1 = P_1$ and $NewP_2 = P_2$ and …)
    then exit
  $P_1 = NewP_1$; $P_2 = NewP_2$; …
Endloop

Example:

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

➔ $T(x,y) :- R(x,y) \cup \Pi_{xy}(R(x,z) \bowtie T(z,y))$

Naïve Datalog Evaluation Algorithm

Datalog program:

P_{i1} :- $body_1$
P_{i2} :- $body_2$
….

➔ Group by IDB predicate

P_1 :- $body_{11} \cup body_{12} \cup \ldots$
P_2 :- $body_{21} \cup body_{22} \cup \ldots$
….

➔ Each rule is a Select-Project-Join-Union query

P_1 :- $SPJU_1$
P_2 :- $SPJU_2$
….

Naïve datalog evaluation algorithm:

$P_1 = P_2 = \ldots = \varnothing$
Loop
  $NewP_1 = SPJU_1$; $NewP_2 = SPJU_2$; …
  if ($NewP_1 = P_1$ and $NewP_2 = P_2$ and …)
    then exit
  $P_1 = NewP_1$; $P_2 = NewP_2$; …
Endloop

Example:

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

➔ $T(x,y) :- R(x,y) \cup \Pi_{xy}(R(x,z) \bowtie T(z,y))$

$T = \varnothing$
Loop
  $NewT(x,y) = R(x,y) \cup \Pi_{xy}(R(x,z) \bowtie T(z,y))$
  if (NewT = T)
    then exit
  T = NewT
Endloop

Discussion

- A datalog program *always* terminates (why?)
- What is the running time of a datalog program as a function of the input database?

Discussion

- A datalog program <u>*always*</u> terminates (why?)
  - Number of possible tuples in IDB is $|Dom|^{arity(R)}$
- What is the running time of a datalog program as a function of the input database?
  - Number of iteration is $\leq |Dom|^{arity(R)}$
  - Each iteration is a relational query

Problem with the Naïve Algorithm

- The same facts are discovered over and over again

- The *semi-naïve* algorithm tries to reduce the number of facts discovered multiple times

Incremental View Maintenance

Let V be a view computed by one datalog rule (no recursion)

V :- body

If (some of) the relations are updated: $R_1 \leftarrow R_1 \cup \Delta R_1$, $R_1 \leftarrow R_2 \cup \Delta R_2$, …

Then the view is also modified as follows: $V \leftarrow V \cup \Delta V$

<u>Incremental view maintenance</u>:
Compute ΔV without having to recompute V

Incremental View Maintenance

Example 1:

$V(x,y)$:- $R(x,z), S(z,y)$

If $R \leftarrow R \cup \Delta R$ then what is $\Delta V(x,y)$?

Incremental View Maintenance

Example 1:

$V(x,y)$:- $R(x,z), S(z,y)$

If $R \leftarrow R \cup \Delta R$ then what is $\Delta V(x,y)$?

$\Delta V(x,y)$:- $\Delta R(x,z), S(z,y)$

Incremental View Maintenance

Example 2:

$V(x,y)$:- $R(x,z), S(z,y)$

If $R \leftarrow R \cup \Delta R$ and $S \leftarrow S \cup \Delta S$
then what is $\Delta V(x,y)$?

Incremental View Maintenance

Example 2:

$V(x,y)$:- $R(x,z), S(z,y)$

If $R \leftarrow R \cup \Delta R$ and $S \leftarrow S \cup \Delta S$
then what is $\Delta V(x,y)$?

$\Delta V(x,y)$:- $\Delta R(x,z), S(z,y)$
$\Delta V(x,y)$:- $R(x,z), \Delta S(z,y)$
$\Delta V(x,y)$:- $\Delta R(x,z), \Delta S(z,y)$

Incremental View Maintenance

Example 3:

$V(x,y)$:- $T(x,z), T(z,y)$

If $T \leftarrow T \cup \Delta T$
then what is $\Delta V(x,y)$?

Incremental View Maintenance

Example 3:

$V(x,y)$:- $T(x,z), T(z,y)$

If $T \leftarrow T \cup \Delta T$
then what is $\Delta V(x,y)$?

$\Delta V(x,y)$:- $\Delta T(x,z), T(z,y)$
$\Delta V(x,y)$:- $T(x,z), \Delta T(z,y)$
$\Delta V(x,y)$:- $\Delta T(x,z), \Delta T(z,y)$

Semi-naïve Evaluation Algorithm

- Naïve algorithm:

P_0 = InitialValue
Repeat
    $P_k = f(P_{k-1})$
Until *no-more-change*

- Semi-naïve algorithm

Semi-naïve Evaluation Algorithm

- Naïve algorithm:

P_0 = InitialValue
Repeat
    $P_k = f(P_{k-1})$
Until *no-more-change*

- Semi-naïve algorithm

$P_0 = \Delta_0$ = InitialValue
Repeat
    $\Delta_k = \Delta f(P_{k-1}, \Delta_{k-1}) - P_{k-1}$
    $P_k = P_{k-1} \cup \Delta_k$
Until *no-more-change*

Semi-naïve Evaluation Algorithm

Separate the Datalog program into the non-recursive, and the recursive part.
Each P_i defined by non-recursive-$SPJU_i$ and (recursive-)$SPJU_i$.

$P_1 = \Delta P_1$ = non-recursive-$SPJU_1$, $P_2 = \Delta P_2$ = non-recursive-$SPJU_2$, …

Loop

$\Delta P_1 = \Delta SPJU_1(P_1, P_2 \ldots, \Delta P_1, \Delta P_2 \ldots) - P_1$;

$\Delta P_2 = \Delta SPJU_2(P_1, P_2 \ldots, \Delta P_1, \Delta P_2 \ldots) - P_2$;

…

if ($\Delta P_1 = \varnothing$ and $\Delta P_2 = \varnothing$ and …)

then break

$P_1 = P_1 \cup \Delta P_1$; $P_2 = P_2 \cup \Delta P_2$; …

Endloop

Semi-naïve Evaluation Algorithm

Separate the Datalog program into the non-recursive, and the recursive part.
Each P_i defined by non-recursive-$SPJU_i$ and (recursive-)$SPJU_i$.

$P_1 = \Delta P_1$ = non-recursive-$SPJU_1$, $P_2 = \Delta P_2$ = non-recursive-$SPJU_2$, …
Loop
    $\Delta P_1 = \Delta SPJU_1(P_1, P_2 \ldots, \Delta P_1, \Delta P_2 \ldots) - P_1$;
    $\Delta P_2 = \Delta SPJU_2(P_1, P_2 \ldots, \Delta P_1, \Delta P_2 \ldots) - P_2$;
    …
    if ($\Delta P_1 = \varnothing$ and $\Delta P_2 = \varnothing$ and …)
        then break
    $P_1 = P_1 \cup \Delta P_1$; $P_2 = P_2 \cup \Delta P_2$; …
Endloop

Example:

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

$T = \Delta T$ = **? (non-recursive rule)**
Loop
    $\Delta T(x,y)$ = **? (recursive Δ-rule)**
    if ($\Delta T = \varnothing$)
        then break
    $T = T \cup \Delta T$
Endloop

Semi-naïve Evaluation Algorithm

Separate the Datalog program into the non-recursive, and the recursive part.
Each P_i defined by non-recursive-$SPJU_i$ and (recursive-)$SPJU_i$.

$P_1 = \Delta P_1$ = non-recursive-$SPJU_1$, $P_2 = \Delta P_2$ = non-recursive-$SPJU_2$, …
Loop
    $\Delta P_1 = \Delta SPJU_1(P_1, P_2 \ldots, \Delta P_1, \Delta P_2 \ldots) - P_1$;
    $\Delta P_2 = \Delta SPJU_2(P_1, P_2 \ldots, \Delta P_1, \Delta P_2 \ldots) - P_2$;
    …
    if ($\Delta P_1 = \varnothing$ and $\Delta P_2 = \varnothing$ and …)
        then break
    $P_1 = P_1 \cup \Delta P_1$; $P_2 = P_2 \cup \Delta P_2$; …
Endloop

Example:

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

T(x,y) = ΔT(x,y) = R(x,y)
Loop
    ΔT(x,y) = R(x,z), ΔT(z,y), not T(x,y)
    if (ΔT = ∅)
        then break
    T = T ∪ ΔT
Endloop

Semi-naïve Evaluation Algorithm

Separate the Datalog program into the non-recursive, and the recursive part.
Each P_i defined by non-recursive-$SPJU_i$ and (recursive-)$SPJU_i$.

$P_1 = \Delta P_1$ = non-recursive-$SPJU_1$, $P_2 = \Delta P_2$ = non-recursive-$SPJU_2$, …
Loop
    $\Delta P_1 = \Delta SPJU_1(P_1, P_2 \ldots, \Delta P_1, \Delta P_2 \ldots) - P_1$;
    $\Delta P_2 = \Delta SPJU_2(P_1, P_2 \ldots, \Delta P_1, \Delta P_2 \ldots) - P_2$;
    …
    if ($\Delta P_1 = \varnothing$ and $\Delta P_2 = \varnothing$ and …)
        then break
    $P_1 = P_1 \cup \Delta P_1$; $P_2 = P_2 \cup \Delta P_2$; …
Endloop

Example:

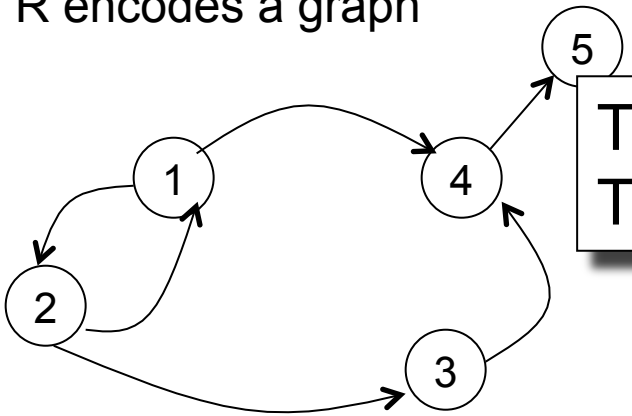
```
T(x,y) :- R(x,y)
T(x,y) :- R(x,z), T(z,y)
```

T(x,y) = ΔT(x,y) = R(x,y)
Loop
    ΔT(x,y) = R(x,z), ΔT(z,y), not T(x,y)
    if (ΔT = ∅)
        then break
    T = T ∪ ΔT
Endloop

Note: for any linear datalog programs, the semi-naïve algorithm has only one Δ-rule for each rule!

Simple datalog programs

R encodes a graph




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

```
T= ΔT = R
Loop
 ΔT(x,y)= R(x,z), ΔT(z,y),not T(x,y)
 if (ΔT = ∅)
       then break
 T = T ∪ ΔT
Endloop
```

R=

1	2
1	4
2	1
2	3
3	4
4	5

Initially:

ΔT=

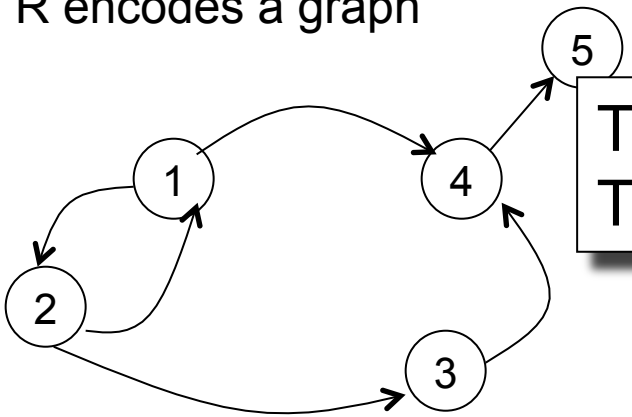
1	2
1	4
2	1
2	3
3	4
4	5

T=

1	2
1	4
2	1
2	3
3	4
4	5

Simple datalog programs

R encodes a graph




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

```
T= ΔT = R
Loop
 ΔT(x,y)= R(x,z), ΔT(z,y),not T(x,y)
 if (ΔT = ∅)
       then break
 T = T ∪ ΔT
Endloop
```

R=

1	2
1	4
2	1
2	3
3	4
4	5

Initially:

ΔT=

1	2
1	4
2	1
2	3
3	4
4	5

T=

1	2
1	4
2	1
2	3
3	4
4	5

First iteration:

ΔT= paths of length 2

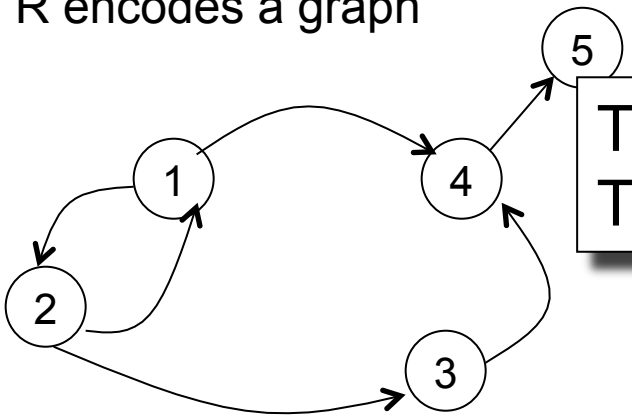
1	1
1	3
1	5
2	2
2	4
3	5

T=

1	2
1	4
2	1
2	3
3	4
4	5
1	1
1	3
1	5
2	2
2	4
3	5

Simple datalog programs

R encodes a graph




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

```
T= ΔT = R
Loop
 ΔT(x,y)= R(x,z), ΔT(z,y),not T(x,y)
 if (ΔT = ∅)
       then break
 T = T ∪ ΔT
Endloop
```

R=

1	2
1	4
2	1
2	3
3	4
4	5

Initially:

ΔT=

1	2
1	4
2	1
2	3
3	4
4	5

T=

1	2
1	4
2	1
2	3
3	4
4	5

First iteration:

ΔT= paths of length 2

1	1
1	3
1	5
2	2
2	4
3	5

T=

1	2
1	4
2	1
2	3
3	4
4	5
1	1
1	3
1	5
2	2
2	4
3	5

Second iteration:

ΔT= paths of length 3

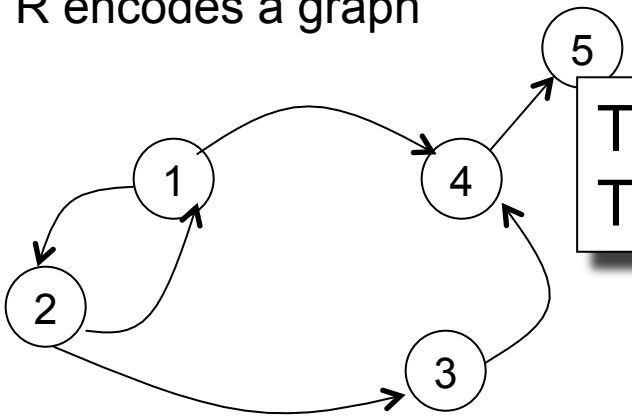
1	2
1	4
2	1
2	3
2	5

T=

1	2
1	4
2	1
2	3
3	4
4	5
1	1
1	3
1	5
2	2
2	4
3	5
2	5

Simple datalog programs

R encodes a graph




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

```
T= ΔT = R
Loop
 ΔT(x,y)= R(x,z), ΔT(z,y),not T(x,y)
 if (ΔT = ∅)
      then break
 T = T ∪ ΔT
Endloop
```

R=

1	2
1	4
2	1
2	3
3	4
4	5

Initially:

ΔT=

1	2
1	4
2	1
2	3
3	4
4	5

T=

1	2
1	4
2	1
2	3
3	4
4	5

First iteration:

ΔT= paths of length 2

1	1
1	3
1	5
2	2
2	4
3	5

T=

1	2
1	4
2	1
2	3
3	4
4	5
1	1
1	3
1	5
2	2
2	4
3	5

Second iteration:

ΔT= paths of length 3

1	2
1	4
2	1
2	3
2	5

T=

1	2
1	4
2	1
2	3
3	4
4	5
1	1
1	3
1	5
2	2
2	4
3	5
2	5

Third iteration:

ΔT= paths of length 4

Discussion of Semi-Naïve Algorithm

- Avoids re-computing some tuples, but not all tuples
- Easy to implement, no disadvantage over naïve
- A rule is called *linear* if its body contains only one recursive IDB predicate:
  - A linear rule always results in a single incremental rule
  - A non-linear rule may result in multiple incremental rules

Summary So Far

- Simple syntax for expressing queries with recursion
- Bottom-up evaluation – always terminates
  - Naïve evaluation
  - Semi-naïve evaluation
- Next:
  - Datalog semantics
  - Datalog with negation

Semantics of a Datalog Program

Three different, equivalent semantics:

- Minimal model semantics
- Least fixpoint semantics
- Proof-theoretic semantics

Minimal Model Semantics

To each rule r: $P(x_1 \ldots x_k)$:- $R_1(\ldots), R_2(\ldots), \ldots$

Minimal Model Semantics

To each rule r: $P(x_1 \ldots x_k)$:- $R_1(\ldots), R_2(\ldots), \ldots$

All variables in the rule

Associate the logical sentence Σ_r: $\forall z_1 \ldots \forall z_n. [(R_1(\ldots) \wedge R_2(\ldots) \wedge \ldots) \rightarrow P(\ldots)]$

Minimal Model Semantics

To each rule r: $P(x_1 \ldots x_k)$:- $R_1(\ldots), R_2(\ldots), \ldots$

All variables in the rule

Associate the logical sentence Σ_r: $\forall z_1 \ldots \forall z_n. [(R_1(\ldots) \wedge R_2(\ldots) \wedge \ldots) \rightarrow P(\ldots)]$

Same as: $\forall x_1 \ldots \forall x_k. [\exists y_1 \ldots \exists y_m.(R_1(\ldots) \wedge R_2(\ldots) \wedge \ldots) \rightarrow P(\ldots)]$

Head variables

Existential variables

Minimal Model Semantics

To each rule r: $P(x_1 \ldots x_k)$:- $R_1(\ldots), R_2(\ldots), \ldots$

All variables in the rule

Associate the logical sentence Σ_r: $\forall z_1 \ldots \forall z_n. [(R_1(\ldots) \wedge R_2(\ldots) \wedge \ldots) \rightarrow P(\ldots)]$

Same as: $\forall x_1 \ldots \forall x_k. [\exists y_1 \ldots \exists y_m.(R_1(\ldots) \wedge R_2(\ldots) \wedge \ldots) \rightarrow P(\ldots)]$

Head variables

Existential variables

<u>Definition</u>. If **P** is a datalog program,
$\Sigma_{\mathbf{P}}$ is the set of all logical sentences associated to its rules.

Minimal Model Semantics

To each rule r: $P(x_1 \ldots x_k)$:- $R_1(\ldots), R_2(\ldots), \ldots$

All variables in the rule

Associate the logical sentence Σ_r: $\forall z_1 \ldots \forall z_n. [(R_1(\ldots) \wedge R_2(\ldots) \wedge \ldots) \rightarrow P(\ldots)]$

Same as: $\forall x_1 \ldots \forall x_k. [\exists y_1 \ldots \exists y_m.(R_1(\ldots) \wedge R_2(\ldots) \wedge \ldots) \rightarrow P(\ldots)]$

Head variables

Existential variables

<u>Definition</u>. If **P** is a datalog program,
$\Sigma_{\mathbf{P}}$ is the set of all logical sentences associated to its rules.

Example. Rule: T(x,y) :- R(x,z), T(z,y)

Sentence: $\forall x. \forall y. \forall z.(R(x,z) \wedge T(z,y) \rightarrow T(x,y))$
$\equiv \forall x. \forall y.(\exists z.R(x,z) \wedge T(z,y) \rightarrow T(x,y))$

Minimal Model Semantics

<u>Definition</u>. A pair (I,J) where I is an EDB and J is an IDB is a *model* for P, if $(I,J) \models \Sigma_{\mathbf{P}}$

<u>Definition</u>. Given an EDB database instance I and a datalog program **P**, the minimal model, denoted J = **P**(I) is a minimal database instance J s.t. $(I,J) \models \Sigma_{\mathbf{P}}$

<u>Theorem</u>. The minimal model always exists, and is unique.

Minimal Model Semantics

<u>Definition</u>. A pair (I,J) where I is an EDB and J is an IDB is a *model* for P, if $(I,J) \models \Sigma_P$

<u>Definition</u>. Given an EDB database instance I and a datalog program **P**, the minimal model, denoted J = **P**(I) is a minimal database instance J s.t. $(I,J) \models \Sigma_P$

<u>Theorem</u>. The minimal model always exists, and is unique.

Example:


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

Which of these IDBs are *<u>models</u>*?
Which are *<u>minimal models</u>*?

R=

1	2
2	3
3	4
4	5

T=

1	2
2	3
3	4
4	5
1	3
2	4
3	5

Minimal Model Semantics

Definition. A pair (I,J) where I is an EDB and J is an IDB is a *model* for P, if $(I,J) \models \Sigma_P$

Definition. Given an EDB database instance I and a datalog program **P**, the minimal model, denoted J = **P**(I) is a minimal database instance J s.t. $(I,J) \models \Sigma_P$

Theorem. The minimal model always exists, and is unique.

Example:

Which of these IDBs are *models*?
Which are *minimal models*?

R=

1	2
2	3
3	4
4	5

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

T=

1	2
2	3
3	4
4	5
1	3
2	4
3	5

T=

1	2
2	3
3	4
4	5
1	3
2	4
3	5
1	4
2	5
1	5

Minimal Model Semantics

<u>Definition</u>. A pair (I,J) where I is an EDB and J is an IDB is a *model* for P, if (I,J) $\models \Sigma_P$

<u>Definition</u>. Given an EDB database instance I and a datalog program **P**, the minimal model, denoted J = **P**(I) is a minimal database instance J s.t. (I,J) $\models \Sigma_P$

<u>Theorem</u>. The minimal model always exists, and is unique.

Example:

Which of these IDBs are *<u>models</u>*?
Which are *<u>minimal models</u>*?

R=

1	2
2	3
3	4
4	5

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

T=

1	2
2	3
3	4
4	5
1	3
2	4
3	5

T=

1	2
2	3
3	4
4	5
1	3
2	4
3	5
1	4
2	5
1	5

T=

1	1
1	2
1	3
1	4
1	5
…	…
…	…
5	4
5	5

All 25 pairs of nodes

Minimal Fixpoint Semantics

<u>Definition</u>. Fix an EDB I, and a datalog program **P**.
The *<u>immediate consequence</u>* operator $T_{\mathbf{P}}$ is defined as follows.
For any IDB J:

$T_{\mathbf{P}}(J)$ = all IDB facts that are immediate consequences from I and J.

<u>Fact</u>. For any datalog program P, the immediate consequence operator is monotone. In other words, if $J_1 \subseteq J_2$ then $T_{\mathbf{P}}(J_1) \subseteq T_{\mathbf{P}}(J_2)$.

Minimal Fixpoint Semantics

<u>Definition</u>. Fix an EDB I, and a datalog program **P**.
The *<u>immediate consequence</u>* operator $T_{\mathbf{P}}$ is defined as follows.
For any IDB J:
$T_{\mathbf{P}}(J)$ = all IDB facts that are immediate consequences from I and J.

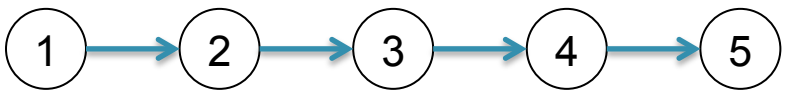
<u>Fact</u>. For any datalog program P, the immediate consequence operator is monotone. In other words, if $J_1 \subseteq J_2$ then $T_{\mathbf{P}}(J_1) \subseteq T_{\mathbf{P}}(J_2)$.

<u>Theorem</u>. The immediate consequence operator has a unique, minimal fixpoint J: $\text{fix}(T_{\mathbf{P}}) = J$, where J is the minimal instance with the property $T_{\mathbf{P}}(J) = J$.

Proof: using Knaster-Tarski's theorem for monotone functions.
The fixpoint is given by:

$$\text{fix}(T_{\mathbf{P}}) = J_0 \cup J_1 \cup J_2 \cup \ldots \quad \text{where } J_0 = \varnothing, \quad J_{k+1} = T_{\mathbf{P}}(J_k)$$

Minimal Fixpoint Semantics



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

R=

1	2
2	3
3	4
4	5

T =

$J_0 = \varnothing$

$J_1 = T_P(J_0)$

1	2
2	3
3	4
4	5

$J_2 = T_P(J_1)$

1	2
2	3
3	4
4	5
1	3
2	4
3	5

$J_3 = T_P(J_2)$

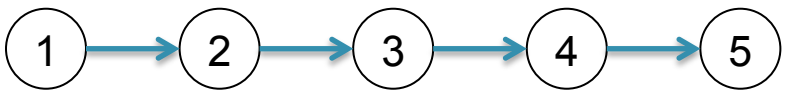
1	2
2	3
3	4
4	5
1	3
2	4
3	5
1	4
2	5

$J_4 = T_P(J_3)$

1	2
2	3
3	4
4	5
1	3
2	4
3	5
1	4
2	5
1	5

Proof Theoretic Semantics

Every fact in the IDB has a *derivation tree*, or *proof tree* justifying its existence.

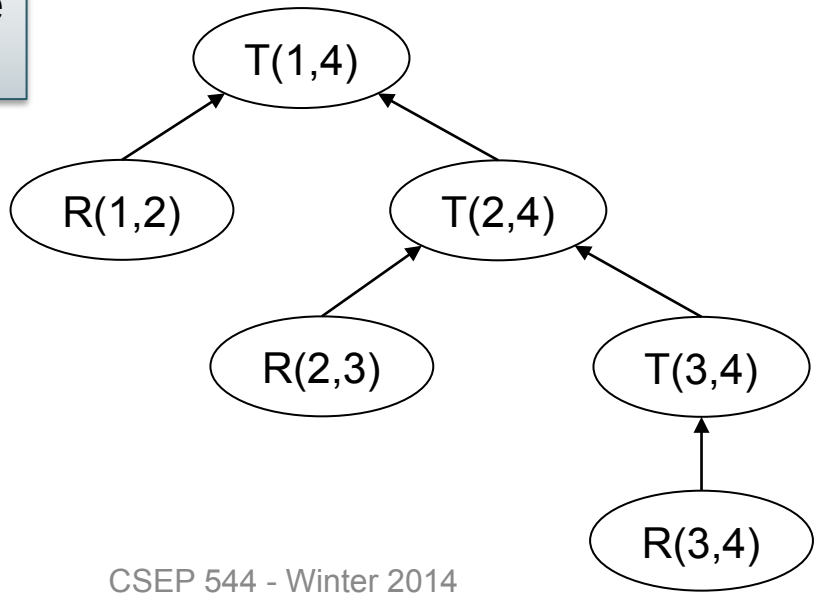


T(x,y) :- R(x,y)
T(x,y) :- R(x,z), T(z,y)

R=

1	2
2	3
3	4
4	5

Derivation tree of T(1,4)



Adding Negation: Datalog¬

Example: compute the complement of the transitive closure

```
T(x,y) :- R(x,y)
T(x,y) :- T(x,z), R(z,y)
CT(x,y) :- Node(x), Node(y), not T(x,y)
```

What does this mean??

Recursion and Negation Don't Like Each Other

EDB: I = { R(a) }

S(x) :- R(x), not T(x)
T(x) :- R(x), not S(x)

Which IDBs are models of **P**?

J_1 = { } J_2 = {S(a)} J_3 = {T(a)} J_4 = {S(a), T(a) }

Recursion and Negation Don't Like Each Other

EDB: I = { R(a) }

S(x) :- R(x), not T(x)
T(x) :- R(x), not S(x)

Which IDBs are models of **P**?

J_1 = { } J_2 = {S(a)} J_3 = {T(a)} J_4 = {S(a), T(a) }

No: both rules fail

Recursion and Negation Don't Like Each Other

EDB: I = { R(a) }

S(x) :- R(x), not T(x)
T(x) :- R(x), not S(x)

Which IDBs are models of **P**?

J_1 = { } J_2 = {S(a)} J_3 = {T(a)} J_4 = {S(a), T(a) }

No: both rules fail

Yes: the facts in J_2 are R(a), S(a), ¬T(a) and both rules are *true*.

Recursion and Negation Don't Like Each Other

EDB: I = { R(a) }

S(x) :- R(x), not T(x)
T(x) :- R(x), not S(x)

Which IDBs are models of **P**?

J_1 = { }

No: both rules fail

J_2 = {S(a)}

Yes: the facts in J_2 are R(a), S(a), ¬T(a) and both rules are *true*.

J_3 = {T(a)}

Yes

J_4 = {S(a), T(a) }

Recursion and Negation Don't Like Each Other

EDB: I = { R(a) }

```
S(x) :- R(x), not T(x)
T(x) :- R(x), not S(x)
```

Which IDBs are models of **P**?

J_1 = { }

No: both rules fail

J_2 = {S(a)}

Yes: the facts in J_2 are R(a), S(a), ¬T(a) and both rules are *true*.

J_3 = {T(a)}

Yes

J_4 = {S(a), T(a) }

Yes

Recursion and Negation Don't Like Each Other

EDB: I = { R(a) }

S(x) :- R(x), not T(x)
T(x) :- R(x), not S(x)

Which IDBs are models of **P**?

J_1 = { }

No: both rules fail

J_2 = {S(a)}

Yes: the facts in J_2 are R(a), S(a), ¬T(a) and both rules are *true*.

J_3 = {T(a)}

Yes

J_4 = {S(a), T(a) }

Yes

There is no *minimal* model!

Recursion and Negation Don't Like Each Other

EDB: I = { R(a) }

S(x) :- R(x), not T(x)
T(x) :- R(x), not S(x)

Which IDBs are models of **P**?

J_1 = { }

No: both rules fail

J_2 = {S(a)}

Yes: the facts in J_2 are R(a), S(a), ¬T(a) and both rules are *true*.

J_3 = {T(a)}

Yes

J_4 = {S(a), T(a) }

Yes

There is no *minimal* model!

There is no minimal fixpoint! (Why does Knaster-Tarski's theorem fail?)

Adding Negation: datalog$^{\neg}$

- **Solution 1: Stratified Datalog$^{\neg}$**
  - Insist that the program be *stratified*: rules are partitioned into strata, and an IDB predicate that occurs only in strata ≤ k may be negated in strata ≥ k+1
- **Solution 2: Inflationary-fixpoint Datalog$^{\neg}$**
  - Compute the fixpoint of $J \cup T_P(J)$
  - Always terminates (why ?)
- **Solution 3: Partial-fixpoint Datalog$^{\neg,*}$**
  - Compute the fixpoint of $T_P(J)$
  - May not terminate

Stratified datalog¬

A datalog¬ program is *stratified* if its rules can be partitioned into k strata, such that:

- If an IDB predicate P appears negated in a rule in stratum i, then it can only appear in the head of a rule in strata 1, 2, …, i-1

P:

P_1 :- $body_1$
P_2 :- $body_2$
….

….

P_j :- $body_j$
….

Stratum i

….

….
P_n :- $body_n$

Note: a datalog¬ program either is stratified or it ain't!

Which programs are stratified?

```
T(x,y) :- R(x,y)
T(x,y) :- T(x,z), R(z,y)
CT(x,y) :- Node(x), Node(y), not T(x,y)
```

```
S(x) :- R(x), not T(x)
T(x) :- R(x), not S(x)
```

Stratified datalog¬

- Evaluation algorithm for stratified datalog¬:

- For each stratum i = 1, 2, …, do:
  - Treat all IDB's defined in prior strata as EBS
  - Evaluate the IDB's defined in stratum i, using either the naïve or the semi-naïve algorithm

Does this compute a minimal model?

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

CT(x,y) :- Node(x), Node(y), not T(x,y)
```

Stratified datalog¬

- Evaluation algorithm for stratified datalog¬:

- For each stratum i = 1, 2, …, do:
  - Treat all IDB's defined in prior strata as EBS
  - Evaluate the IDB's defined in stratum i, using either the naïve or the semi-naïve algorithm

Does this compute a minimal model?

NO:
J_1 = { T = transitive closure, CT = its complement}
J_2 = { T = all pairs of nodes, CT = empty}

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

CT(x,y) :- Node(x), Node(y), not T(x,y)
```

Inflationary-fixpoint datalog¬

Let **P** be any datalog¬ program, and I an EDB.
Let $T_{\mathbf{P}}(J)$ be the *immediate consequence* operator.
Let $F(J) = J \cup T_{\mathbf{P}}(J)$ be the *inflationary immediate consequence* operator.

Define the sequence: $J_0 = \varnothing$, $J_{n+1} = F(J_n)$, for $n \geq 0$.

Definition. The inflationary fixpoint semantics of **P** is $J = J_n$ where n is such that $J_{n+1} = J_n$

Why does there always exists an n such that $J_n = F(J_n)$?

Find the inflationary semantics for:

```
T(x,y) :- R(x,y)
T(x,y) :- T(x,z), R(z,y)
CT(x,y) :- Node(x), Node(y), not T(x,y)
```

```
S(x) :- R(x), not T(x)
T(x) :- R(x), not S(x)
```

Inflationary-fixpoint datalog¬

- Evaluation for Inflationary-fixpoint datalog¬
- Use the naïve, of the semi-naïve algorithm
- Inhibit any optimization that rely on monotonicity (e.g. out of order execution)

Partial-fixpoint datalog$^{\neg,*}$

Let **P** be any datalog$^{\neg}$ program, and I an EDB.
Let $T_{\mathbf{P}}(J)$ be the *immediate consequence* operator.

Define the sequence: $J_0 = \varnothing$, $J_{n+1} = T_{\mathbf{P}}(J_n)$, for $n \geq 0$.

Definition. The partial fixpoint semantics of **P** is $J = J_n$ where n is such that $J_{n+1} = J_n$, if such an n exists, undefined otherwise.

Find the partial fixpoint semantics for:

```
T(x,y) :- R(x,y)
T(x,y) :- T(x,z), R(z,y)
CT(x,y) :- Node(x), Node(y), not T(x,y)
```

Note: there may not exists an n such that $J_n = F(J_n)$

```
S(x) :- R(x), not T(x)
T(x) :- R(x), not S(x)
```

Discussion

- Which semantics does Daedalus adopt?

Discussion

- Which semantics does Daedalus adopt?
- A: stratified semantics, but with a twist: every timestamp is a different strata

Discussion

- Which semantics does Daedalus adopt?
- A: stratified semantics, but with a twist: every timestamp is a different strata
- E.g. negation of transitive closure:

```
T(x,y) :- R(x,y)
T(x,y) :- T(x,z), R(z,y)
CT(x,y)@next :- Node(x), Node(y), not T(x,y)
```

Discussion

Comparing datalog¬

- Compute the complement of the transitive closure in inflationary datalog¬
- Compare the expressive power of:
  - Stratified datalog¬
  - Inflationary fixpoint datalog¬
  - Partial fixpoint datalog¬

Data Provenance

Data Provenance

- Provenance inside the DBMS
  - Will discuss today
- Provenance outside of the DBMS
  - Much more messy; there is a standard, OPM (Open Provenance Model)

Provenance Annotations

- Some query produces an output table T(A,B,C)
- We store it over some period of time
- Later we ask: “where did this tuple come from?”
- The “provenance annotation” answers this.

A	B	C	
a1	b1	c1	provenance1
a2	b1	c1	provenance2
a2	b2	c2	provenance3
a2	b2	c3	provenance4

Provenance Annotations

- Start by annotating each tuple in the original database with a unique identifier; can be the Tuple Id (TID)

A	B	
a1	b1	X1
a2	b1	X2
a2	b2	X3

- Next, compute the provenance expression inductively, based on the query plan

Join Operator

$\bowtie$

A	B	
a1	b1	X1
a2	b1	X2
a2	b2	X3

B	C	
b1	c1	Y1
b2	c2	Y2
b2	c3	Y3

=

A	B	C	
a1	b1	c1	$X1 \cdot Y1$
a2	b1	c1	$X2 \cdot Y1$
a2	b2	c2	$X3 \cdot Y2$
a2	b2	c3	$X3 \cdot Y3$

Projection Operator

Π

A	B	
a1	b1	X1
a1	b2	X2
a2	b1	X3
a2	b2	X4
a2	b3	X5

=

A	
a1	X1+X2
a2	X3+X4+X5

Union Operator

∪

A	B	
a1	b1	X1
a2	b2	X2

A	B	
a2	b2	Y1
a3	b3	Y2

=

A	B	
a1	b1	X1
a2	b2	X2+Y1
a3	b3	X3

Selection Operator

$\sigma_{A=a1}$

A	B	
a1	b1	X1
a1	b2	X2
a2	b1	X3
a2	b2	X4
a2	b3	X5

=

A	B	
a1	b1	X1
a1	b2	X2

We could simply remove the tuples filtered out.
But it's better to keep them around (we'll see why).
What is their annotation?

Selection Operator

$\sigma_{A=a1}$

A	B	
a1	b1	X1
a1	b2	X2
a2	b1	X3
a2	b2	X4
a2	b3	X5

=

A	B	
a1	b1	$X1 \cdot 1$
a1	b2	$X2 \cdot 1$
a2	b1	$X3 \cdot 0$
a2	b2	$X4 \cdot 0$
a2	b3	$X5 \cdot 0$

We could simply remove the tuples filtered out.
But it's better to keep them around (we'll see why).
What is their annotation?

Complex Example

$$\sigma_{C=e}\Pi_{AC}(\ \Pi_{AC}(R) \bowtie \Pi_{BC}(R) \cup \Pi_{AB}(R) \bowtie \Pi_{BC}(R)) =$$

R =

A	B	C	
a	b	c	X
d	b	e	Y
f	g	e	Z

A	C	
a	c	$(X \cdot X + X \cdot X) \cdot 0 = 2 \cdot X^2$
a	e	$X \cdot Y \cdot 1 = X \cdot Y$
d	c	$Y \cdot X \cdot 0 = 0$
d	e	$(Y \cdot Y + Y \cdot Z + Y \cdot Y) \cdot 1 = 2 \cdot Y^2 + Y \cdot Z$
f	e	$(Z \cdot Z + Z \cdot Y + Z \cdot Z) \cdot 1 = 2 \cdot Z^2 + Y \cdot Z$

Discuss in class what these annotations mean

K-Relations

<u>Definition</u>. A K-relation is a relation where each tuple is annotated with an element from the set K.

What we have described so far is an extension of the positive operations of the relational algebra to K-relations

We assumed that K has the operators +, $\cdot$

Identities on Provenance Expressions

The problem:

- We have defined provenance for a query plan P
- Given a query Q, we want the provenance to be independent of the plan
- Needed: if P1=P2,
  then provenance(P1) = Provenance(P2)

Identities on Provenance Expressions

Definition. A structure $(K, +, \cdot, 0, 1)$ is called a commutative semiring if:

1. $(K,+,0)$ is a commutative monoid:
   a. $+$ is associative: $(x+y)+z=x+(y+z)$
   b. $+$ is commutative: $x+y=y+x$
   c. 0 is the identity for $+$: $x+0=0+x=x$

2. $(K, \cdot, 1)$ is a commutative monoid:
   a. … (similar identities)

3. $\cdot$ distributes over $+$: $x \cdot (y+z) = x \cdot y + x \cdot z$

4. For all x: $x \cdot 0 = 0 \cdot x = 0$

Identities on Provenance Expressions

Definition. A structure $(K, +, \cdot, 0, 1)$ is called a commutative semiring if:

1. $(K,+,0)$ is a commutative monoid:
   a. $+$ is associative: $(x+y)+z=x+(y+z)$
   b. $+$ is commutative: $x+y=y+x$
   c. 0 is the identity for $+$: $x+0=0+x=x$

2. $(K, \cdot, 1)$ is a commutative monoid:
   a. … (similar identities)

3. $\cdot$ distributes over $+$: $x \cdot (y+z) = x \cdot y + x \cdot z$

4. For all x: $x \cdot 0 = 0 \cdot x = 0$

Theorem. The standard identities of the Bag algebra hold for K-relations iff $(K, +, \cdot, 0, 1)$ is a commutative semiring.

Identities on Provenance Expressions

Discuss in class:

$$q(x,u) := R(x,y), S(y,z), T(z,u)$$

Given two plans, why are the annotations equal?

Applications

$$\sigma_{C=e}\Pi_{AC}(\ \Pi_{AC}(R) \bowtie \Pi_{BC}(R) \cup \Pi_{AB}(R) \bowtie \Pi_{BC}(R)) =$$

R =

A	B	C	
a	b	c	X
d	b	e	Y
f	g	e	Z

A	C	
a	c	$2 \cdot X^2$
a	e	$X \cdot Y$
d	e	$2 \cdot Y^2 + Y \cdot Z$
f	e	$2 \cdot Z^2 + Y \cdot Z$

Q: Suppose we delete the tuple (d,b,e) from R. Which tuple(s) disappear from the result?

Applications

$$\sigma_{C=e}\Pi_{AC}(\ \Pi_{AC}(R) \bowtie \Pi_{BC}(R) \cup \Pi_{AB}(R) \bowtie \Pi_{BC}(R)) =$$

R =

A	B	C	
a	b	c	X
d	b	e	Y
f	g	e	Z

A	C	
a	c	$2 \cdot X^2$
a	e	$X \cdot Y$
d	e	$2 \cdot Y^2 + Y \cdot Z$
f	e	$2 \cdot Z^2 + Y \cdot Z$

=

A	C	
a	c	$2 \cdot X^2$
a	e	0
d	e	0
f	e	$2 \cdot Z^2$

Q: Suppose we delete the tuple (d,b,e) from R. Which tuple(s) disappear from the result?

A: Set Y=0

Applications

$$\sigma_{C=e}\Pi_{AC}(\ \Pi_{AC}(R) \bowtie \Pi_{BC}(R) \cup \Pi_{AB}(R) \bowtie \Pi_{BC}(R)) =$$

R =

A	B	C	
a	b	c	X
d	b	e	Y
f	g	e	Z

A	C	
a	c	$2 \cdot X^2$
a	e	$X \cdot Y$
d	e	$2 \cdot Y^2 + Y \cdot Z$
f	e	$2 \cdot Z^2 + Y \cdot Z$

Q: Suppose each tuple in R occurs 3 times (bag semantics). How many times occurs each tuple in the answer?

Applications

$$\sigma_{C=e}\Pi_{AC}(\ \Pi_{AC}(R) \bowtie \Pi_{BC}(R) \cup \Pi_{AB}(R) \bowtie \Pi_{BC}(R)) =$$

R =

A	B	C	
a	b	c	X
d	b	e	Y
f	g	e	Z

A	C	
a	c	$2 \cdot X^2$
a	e	$X \cdot Y$
d	e	$2 \cdot Y^2 + Y \cdot Z$
f	e	$2 \cdot Z^2 + Y \cdot Z$

A	C	
a	c	18
a	e	9
d	e	27
f	e	27

Q: Suppose each tuple in R occurs 3 times (bag semantics). How many times occurs each tuple in the answer?

A. Set X=Y=Z=3

Sets of Contributing Tuples

$$\sigma_{C=e}\Pi_{AC}(\ \Pi_{AC}(R) \bowtie \Pi_{BC}(R) \cup \Pi_{AB}(R) \bowtie \Pi_{BC}(R)) =$$

R =

A	B	C	
a	b	c	X
d	b	e	Y
f	g	e	Z

A	C	
a	c	$2 \cdot X^2$
a	e	$X \cdot Y$
d	e	$2 \cdot Y^2 + Y \cdot Z$
f	e	$2 \cdot Z^2 + Y \cdot Z$

➔

A	C	
a	c	X
a	e	X, Y
d	e	Y, Z
f	e	Y, Z

Trace only the set of input tuples that contributed to an output tuple

This is also a semi-ring! Which one?

Semirings for various models of provenance (1)

R =

A	B	C	
a	b	c	X
d	b	e	Y
f	g	e	Z

Q =

A	C	
d	e	Y,Z

Lineage [CuiWidomWiener 00 etc.]

Sets of contributing tuples

Semiring: $(\mathrm{Lin}(X), +, \cup, \perp, \varnothing)$

Source: Green ICDT 2009

Semirings for various models of provenance (2)

R =

A	B	C	
a	b	c	X
d	b	e	Y
f	g	e	Z

Q =

A	C	
d	e	{{Y},{Y,Z}}

(Witness, Proof) **why-provenance**
[BunemanKhannaTan 01] & [Buneman+ PODS08]

Sets of witnesses (w. =set of contributing tuples)

Semiring: $(\mathrm{Why}(X), \cup, \uplus, \varnothing, \{\varnothing\})$

Source: Tannen, EDBT 2010

Semirings for various models of provenance (3)

R =

A	B	C	
a	b	c	X
d	b	e	Y
f	g	e	Z

Q =

A	C	
d	e	{Y}

Minimal witness **why-provenance**
[BunemanKhannaTan 01]

Sets of minimal witnesses

Semiring: (PosBool(X), $\wedge$, $\vee$, $\top$, $\bot$)

Source: Tannen, EDBT 2010

Semirings for various models of provenance (4)

R =

A	B	C	
a	b	c	X
d	b	e	Y
f	g	e	Z

Q =

A	C	
d	e	[{Y}, {Y}, {Y,Z}]

Notation:

{ } set

[ ] bag

Trio lineage [Das Sarma+ 08]

Bags of sets of contributing tuples (of witnesses)

Semiring: (Trio(X), +, $\cdot$, 0, 1) (defined in [Green, ICDT 09])

Source: Tannen, EDBT 2010

Semirings for various models of provenance (5)

R =

A	B	C	
a	b	c	X
d	b	e	Y
f	g	e	Z

Q =

A	C	
d	e	{[Y,Y], [Y,Z]}

Notation:

{ } set

[ ] bag

Polynomials with boolean coefficients [Green, ICDT 09]

(B[*X*]-provenance)

Sets of bags of contributing tuples

Semiring: (B[*X*], +, ·, 0, 1)

Source: Tannen, EDBT 2010

Semirings for various models of provenance (6)

R =

A	B	C	
a	b	c	X
d	b	e	Y
f	g	e	Z

Q =

A	C	
d	e	[[Y,Y], [Y,Y], [Y,Z]]

Provenance polynomials [GKT, PODS 07]

(N[*X*]-provenance)

Bags of bags of contributing tuples

Semiring: (N[*X*], +, ·, 0, 1)

Source: Tannen, EDBT 2010

Application

Discretionary Access Control [LaPadula]

- Public = P
- Confidential = C
- Secret = S
- Top Secret = T
- No Such Thing… = 0

R =

A	B	C
a	b	c
d	b	e
f	g	e

X=C

Y=P

Z=T

A	C	
a	c	$2 \cdot X^2 = ?$
a	e	$X \cdot Y = ?$
d	e	$2 \cdot Y^2 + Y \cdot Z = ?$
f	e	$2 \cdot Z^2 + Y \cdot Z = ?$

Application

Discretionary Access Control [LaPadula]

- Public = P
- Confidential = C
- Secret = S
- Top Secret = T
- No Such Thing… = 0

R =

A	B	C
a	b	c
d	b	e
f	g	e

X=C

Y=P

Z=T

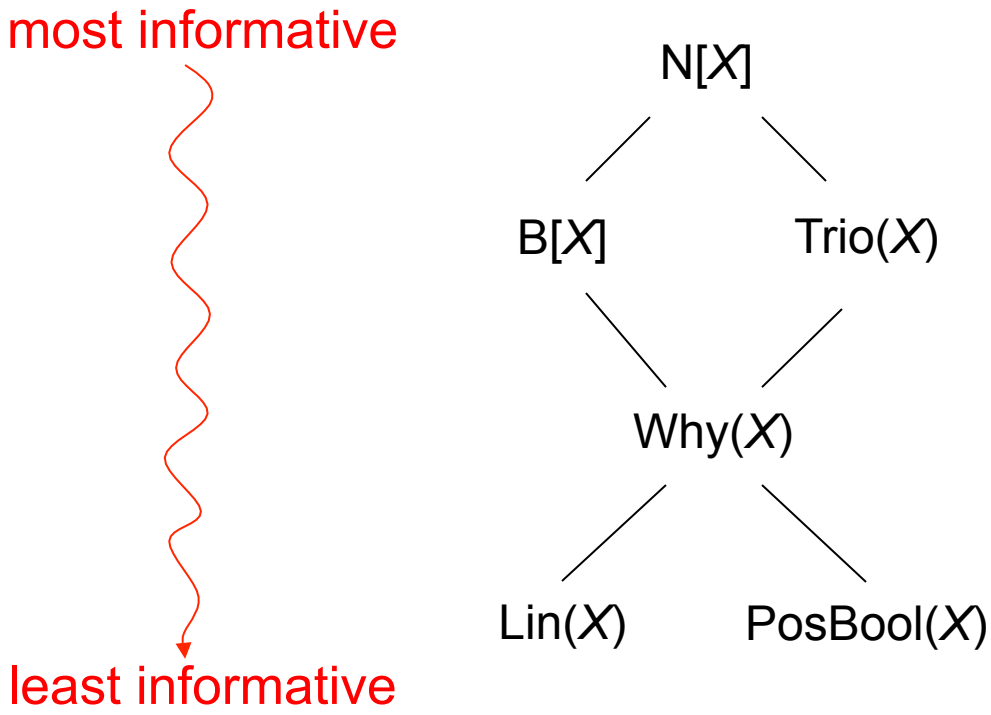
A	C	
a	c	$2 \cdot X^2 = C$
a	e	$X \cdot Y = C$
d	e	$2 \cdot Y^2 + Y \cdot Z = C$
f	e	$2 \cdot Z^2 + Y \cdot Z = T$

(A, min, max, 0, P), where A = P < C < S < T < 0

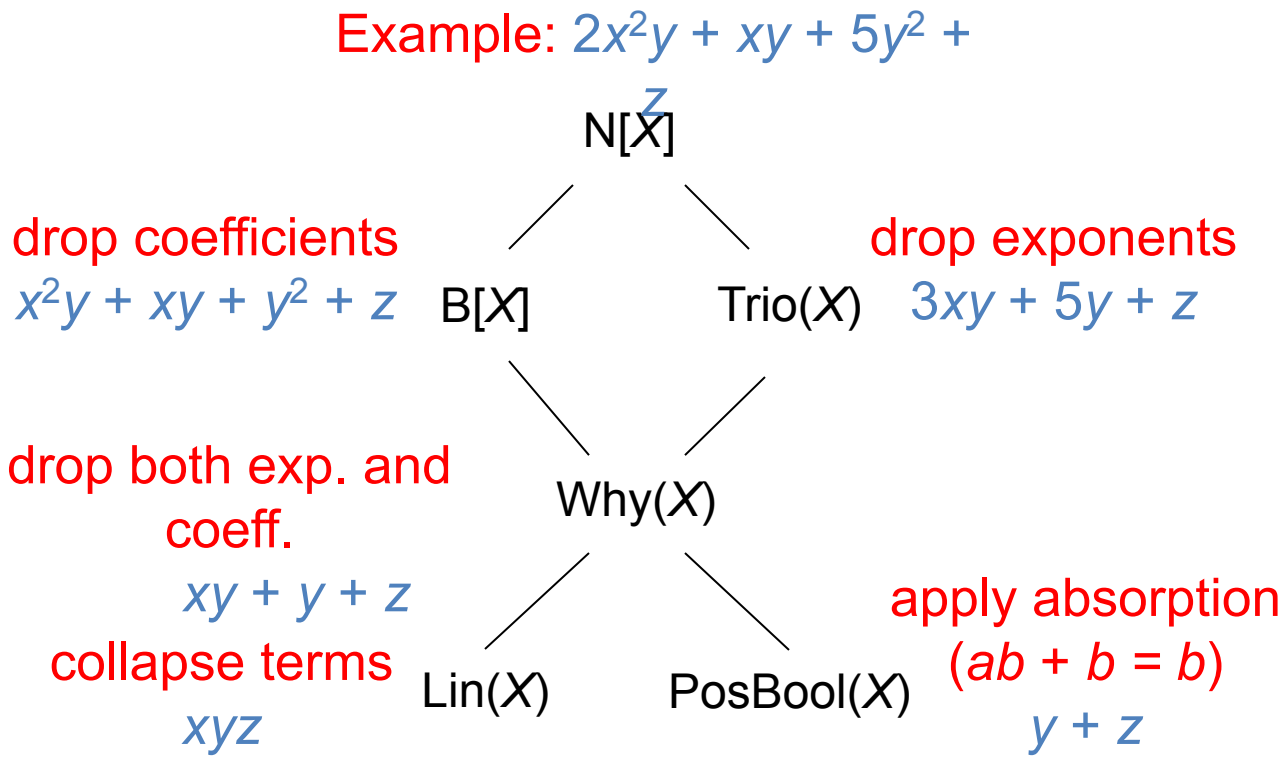
But are there useful commutative semirings?

$(B, \wedge, \vee, \top, \bot)$	Set semantics
$(\mathbb{N}, +, \cdot, 0, 1)$	Bag semantics
$(P(\Omega), \cup, \cap, \varnothing, \Omega)$	Probabilistic events [FuhrRölleke 97]
$(BoolExp(X), \wedge, \vee, \top, \bot)$	Conditional tables (c-tables) [ImielinskiLipski 84]
$(R_+^\infty, \min, +, 1, 0)$	Tropical semiring (cost/distrust score/confidence need)
$(A, \min, \max, 0, P)$ where $A = P < C < S < T < 0$	Access control levels [PODS8]

A provenance hierarchy



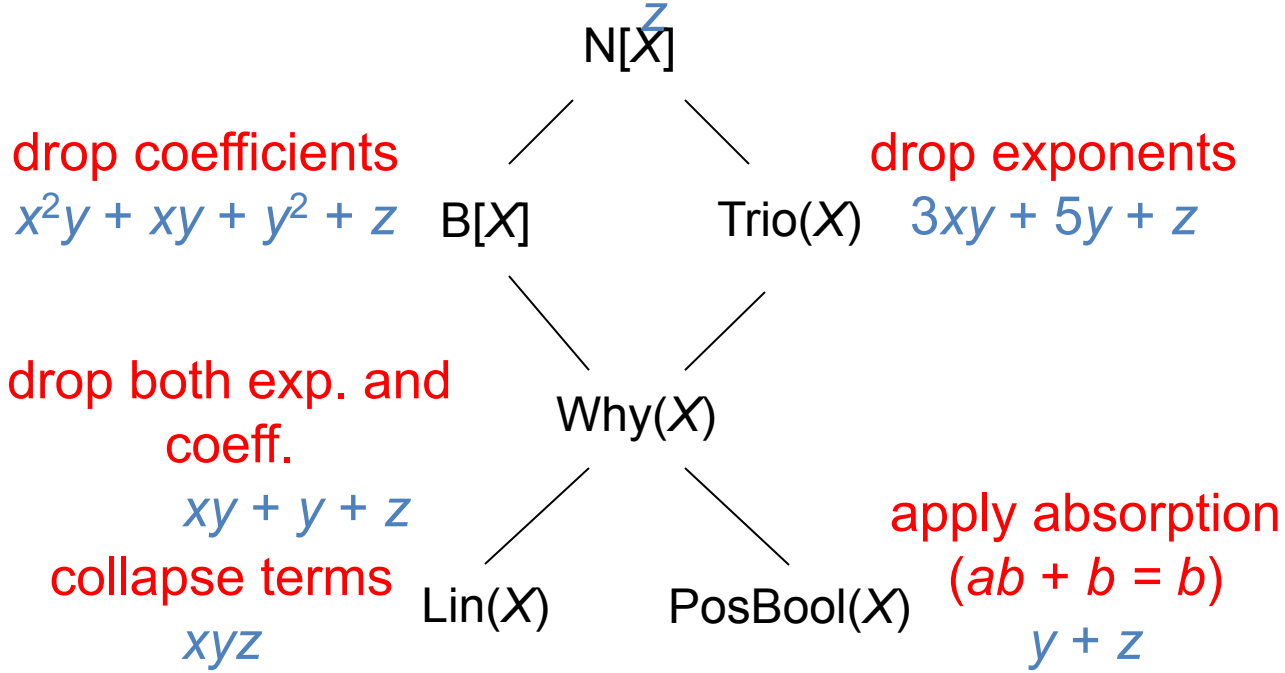
One semiring to rule them all… (apologies!)



A path downward from K_1 to K_2 indicates that there exists an **onto** (**surjective**) **semiring homomorphism** $h : K_1 \rightarrow K_2$

Using homomorphisms to relate models

Example: $2x^2y + xy + 5y^2 + z$



Homomorphism?

$h(x+y) = h(x)+h(y)$ $\quad h(xy)=h(x)h(y)$ $\quad h(0)=0$ $\quad h(1)=1$

Moreover, for these homomorphisms $h(x)= x$