CSE544 Data Management

Lectures 14

Datalog

Announcement

- Project Milestone due today
- HW3 extended to Thursday, 2/29
- HW4 posted, due Monday, 3/11
- Lecture Mo, 3/4 canceled; OH instead
  R4 deadline extended to 3/10

Project

- Project meetings w/ Dan: Friday, 3/1
- Printing the poster:
  - Kyle can help on Monday, 3/4, OR ask a colleague with a cse account
- Poster presentations: Wed, 3/6, 10-2pm
  - In the atrium of Allen building
  - Setup: 9:30; poster + demo (optional)
  - Snacks, pizza will be provided

Recap

```
R1(args) :- body1
R2(args) :- body2
…
```

- Datalog program is a set of rules
- Head, body, atoms
- Head variables, existential variables
- Extensional Database Predicates, EDBs
  Intensional Database Predicates, IDBs

Recap

- Naïve algorithm:
  $\emptyset = IDB_0 \subseteq IDB_1 \subseteq IDB_2 \subseteq \ldots$

- It always terminates, even where other languages diverge:

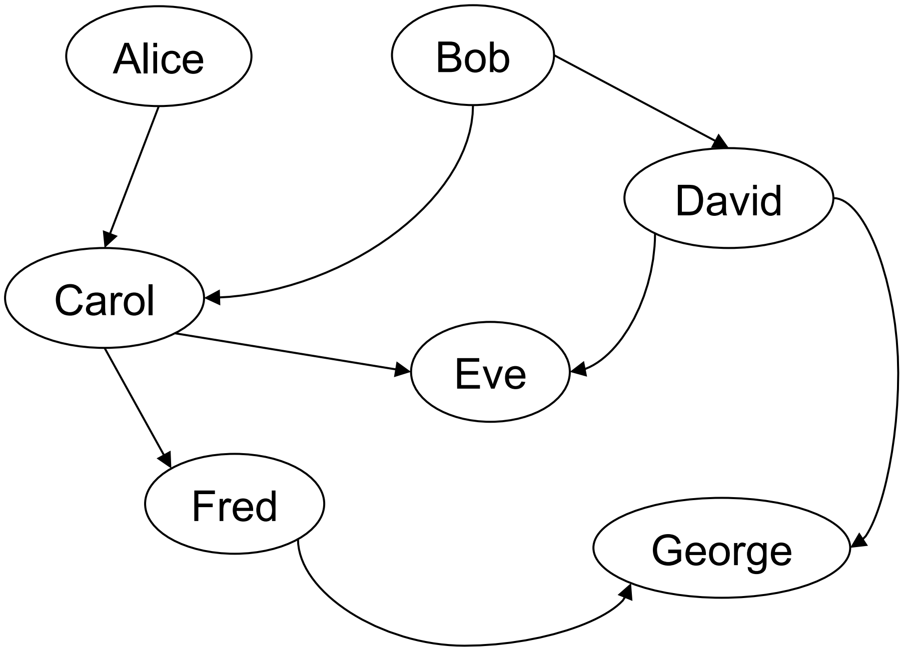
```
T(x,y) :- R(x,y)
T(x,y) :- T(x,z), R(z,y)
```

- However: monotone queries only, no arithmetic x+y etc

Outline

- Examples
- Semi-naïve Evaluation
- Incremental View Maintenance

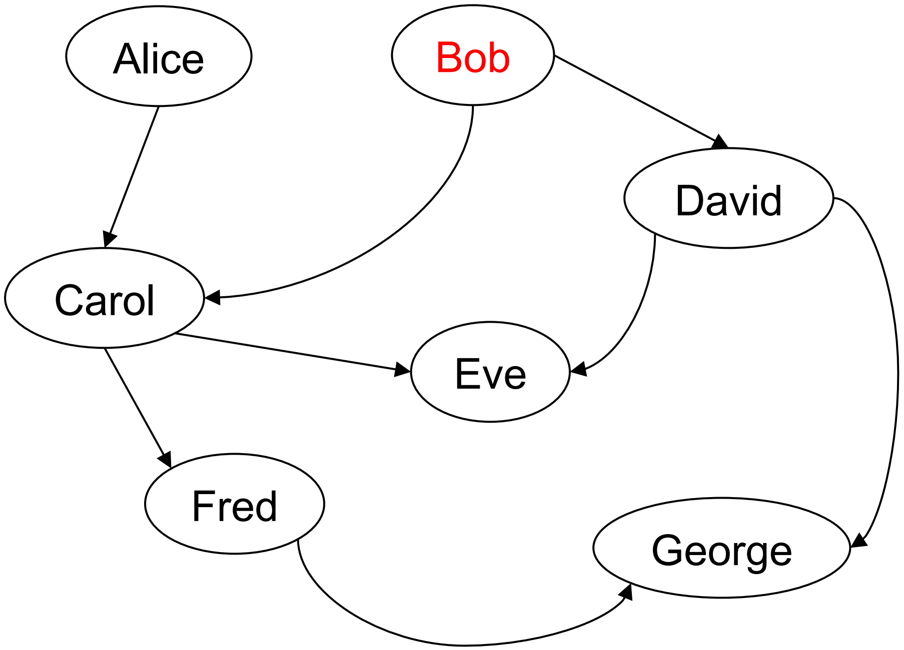
Example: Descendants



Example: Descendants

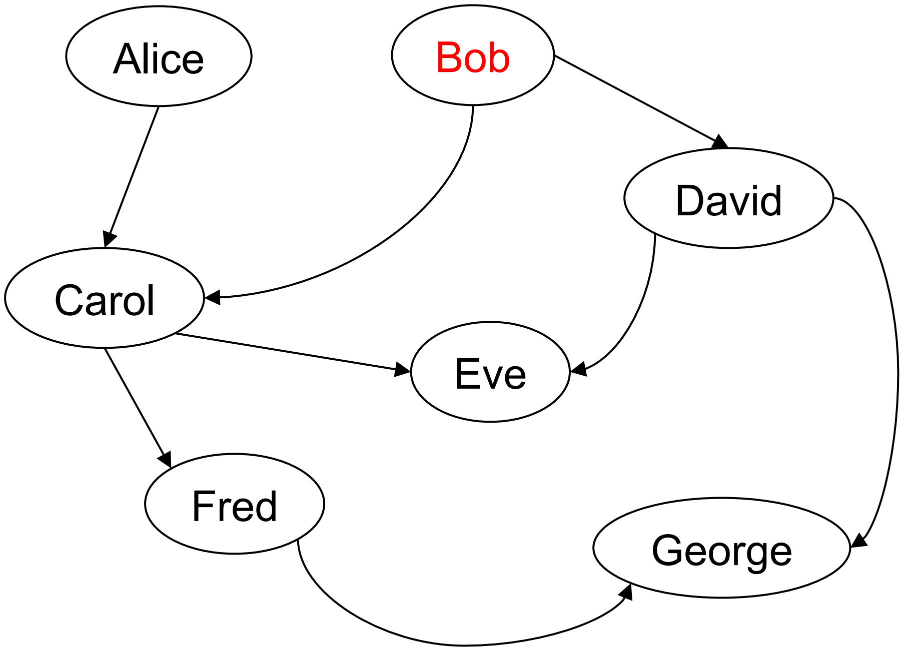
Find all descendants of Bob

D(y) :- Child('Bob',y)
D(y) :- D(x), Child(x,y)



Example: Descendants

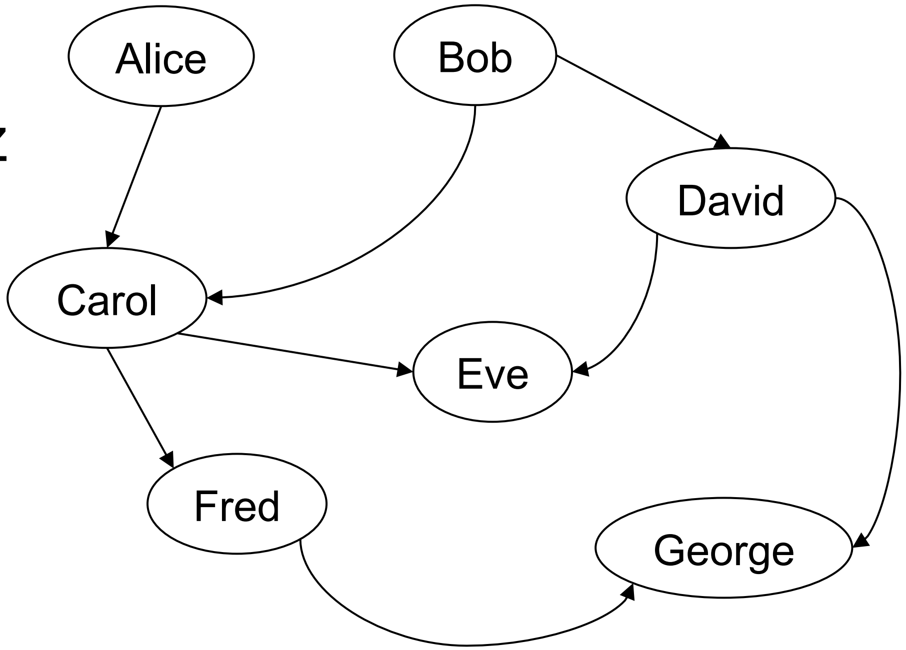
Find all descendants of Bob



Example: Same Generation

x,y are at the same generation

if they have a common ancestor z at the same distance.

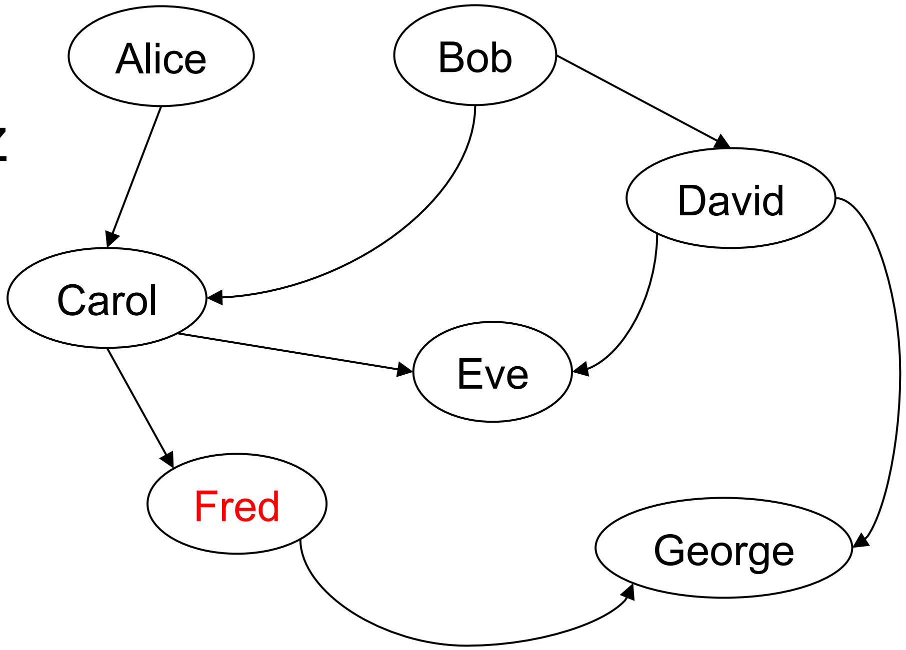


Example: Same Generation

x,y are at the same generation

if they have a common ancestor z at the same distance.

Find people at the same generation as Fred

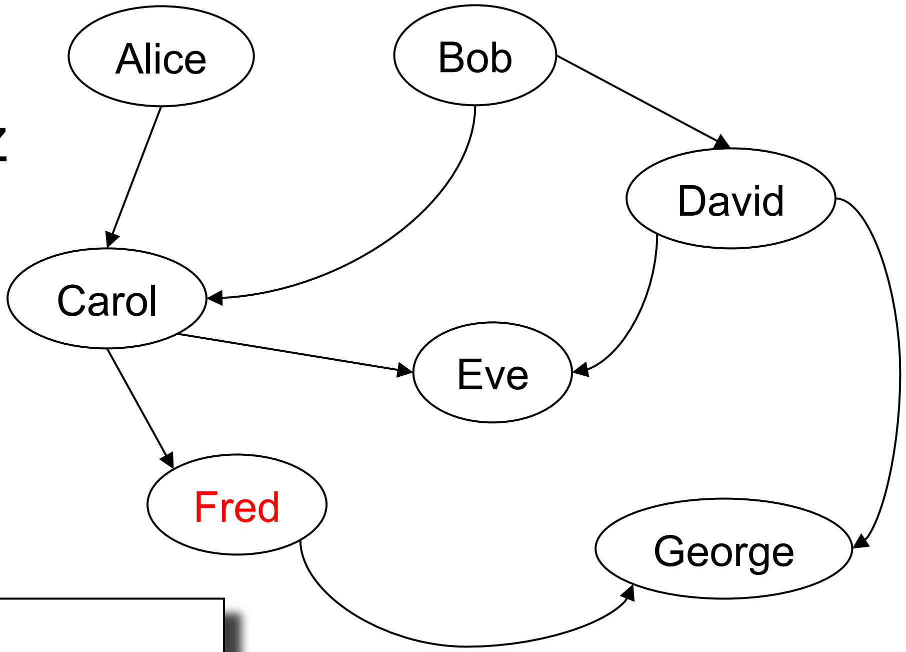


Example: Same Generation

x,y are at the same generation

if they have a common ancestor z at the same distance.

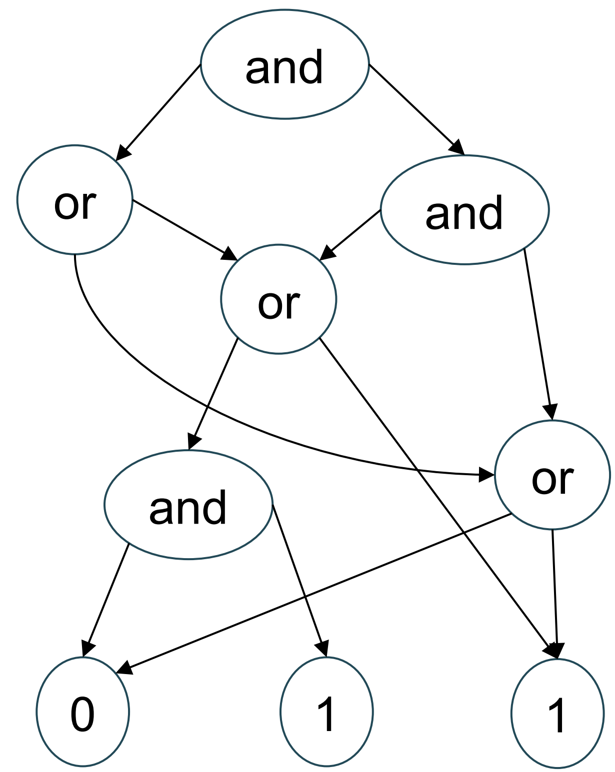
Find people at the same generation as Fred




```
SG(x,y) :- Child(z,x), Child(z,y)
SG(x,y) :- SG(u,v), Child(u,x), Child(v,y)
Answer(y) :- SG('Fred',y)
```

Example: Boolean Circuits

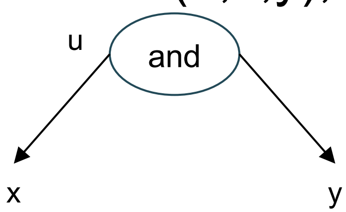
Find all gates whose value is 1

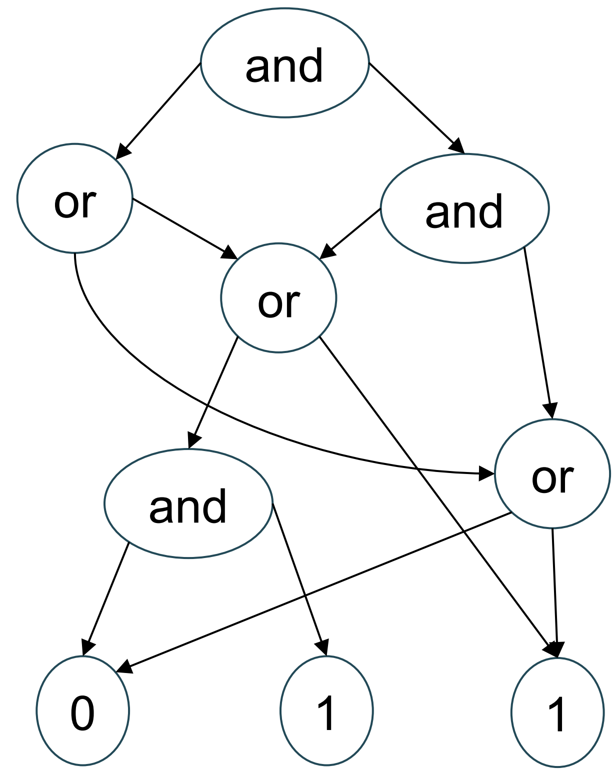


Example: Boolean Circuits

Find all gates whose value is 1

Schema: AND(u,x,y), OR(u,x,y), LEAF(u,0/1)

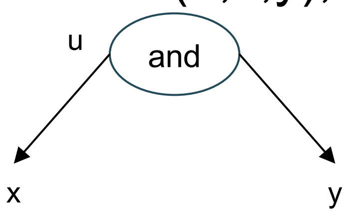


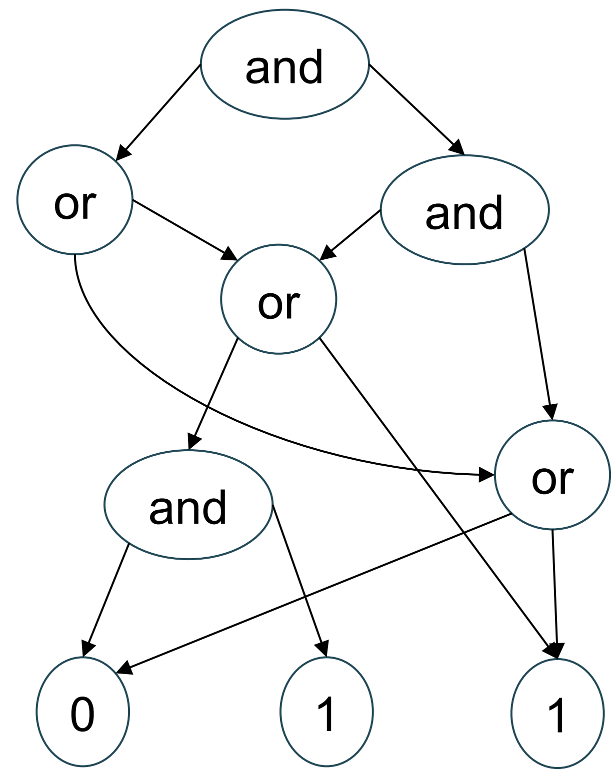


Example: Boolean Circuits

Find all gates whose value is 1

Schema: AND(u,x,y), OR(u,x,y), LEAF(u,0/1)



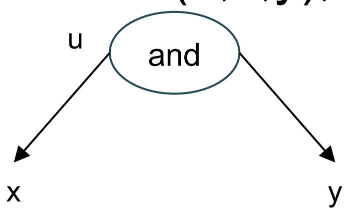


ONE(u) :- LEAF(u,1)

Example: Boolean Circuits

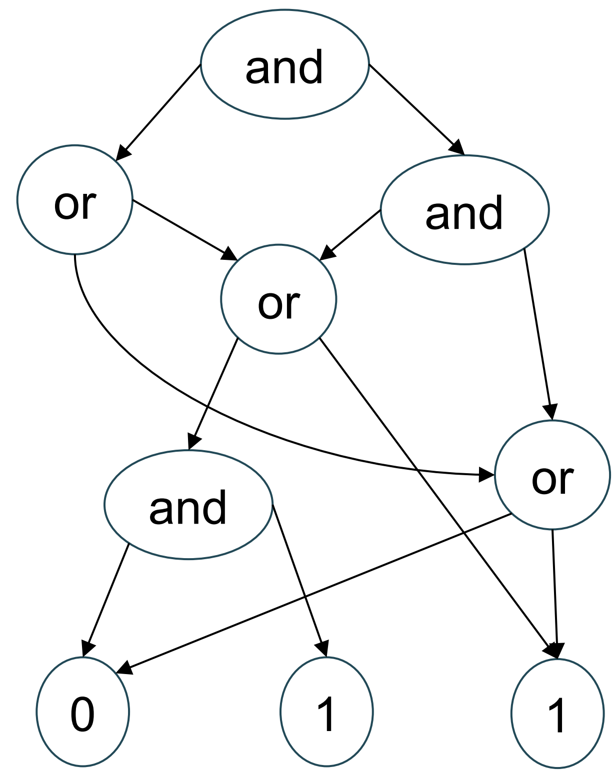
Find all gates whose value is 1

Schema: AND(u,x,y), OR(u,x,y), LEAF(u,0/1)



ONE(u) :- LEAF(u,1)

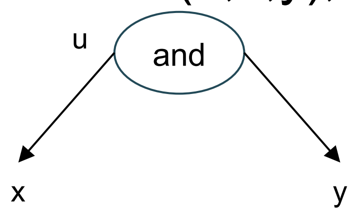
ONE(u) :- AND(u,x,y),ONE(x),ONE(y)

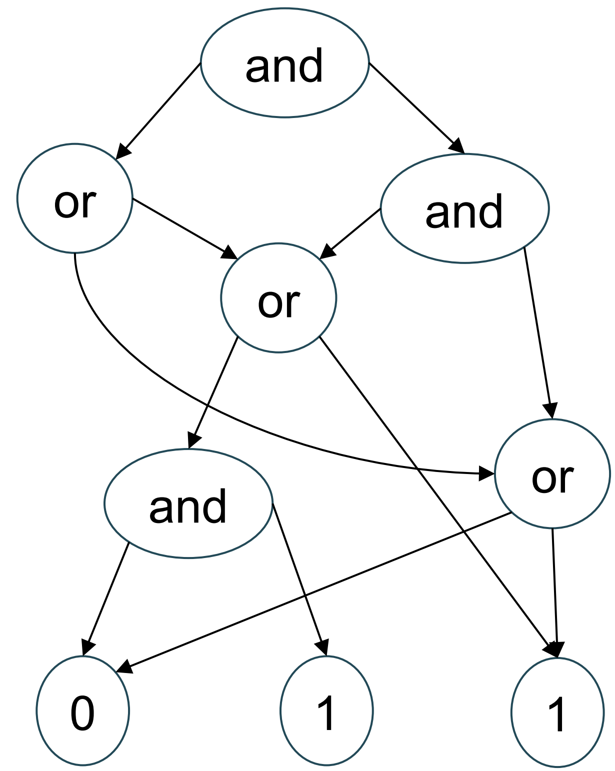


Example: Boolean Circuits

Find all gates whose value is 1

Schema: AND(u,x,y), OR(u,x,y), LEAF(u,0/1)





ONE(u) :- LEAF(u,1)

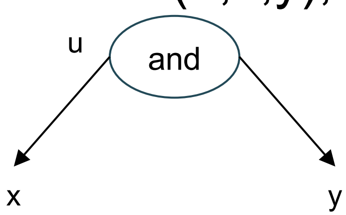
ONE(u) :- AND(u,x,y),ONE(x),ONE(y)

ONE(u) :- OR(u,x,y),ONE(x)

Example: Boolean Circuits

Find all gates whose value is 1

Schema: AND(u,x,y), OR(u,x,y), LEAF(u,0/1)

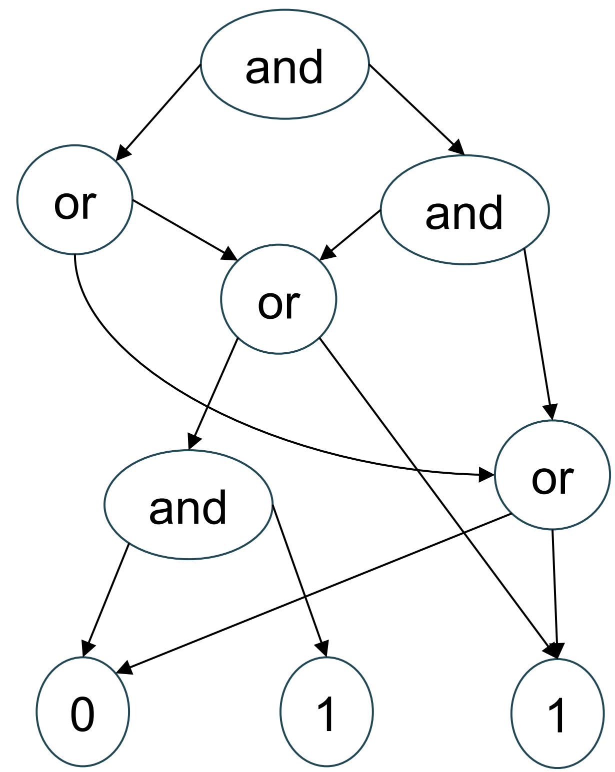


ONE(u) :- LEAF(u,1)

ONE(u) :- AND(u,x,y),ONE(x),ONE(y)

ONE(u) :- OR(u,x,y),ONE(x)

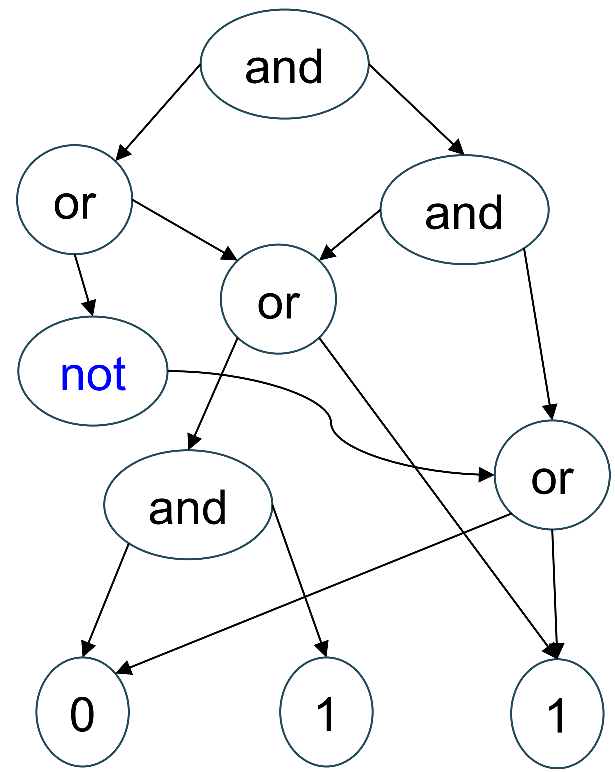
ONE(u) :- OR(u,x,y),ONE(y)



Example: Boolean Circuits

Find all gates whose value is 1

Schema: AND(u,x,y), OR(u,x,y), LEAF(u,0/1)
NOT(u,x)

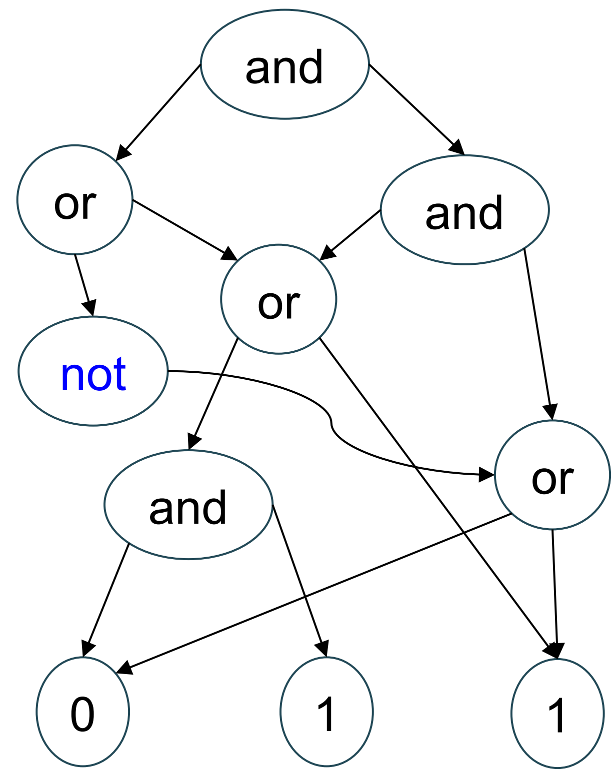


Example: Boolean Circuits

Find all gates whose value is 1

Schema: AND(u,x,y), OR(u,x,y), LEAF(u,0/1)
NOT(u,x)

Problem: we cannot use negation, or difference

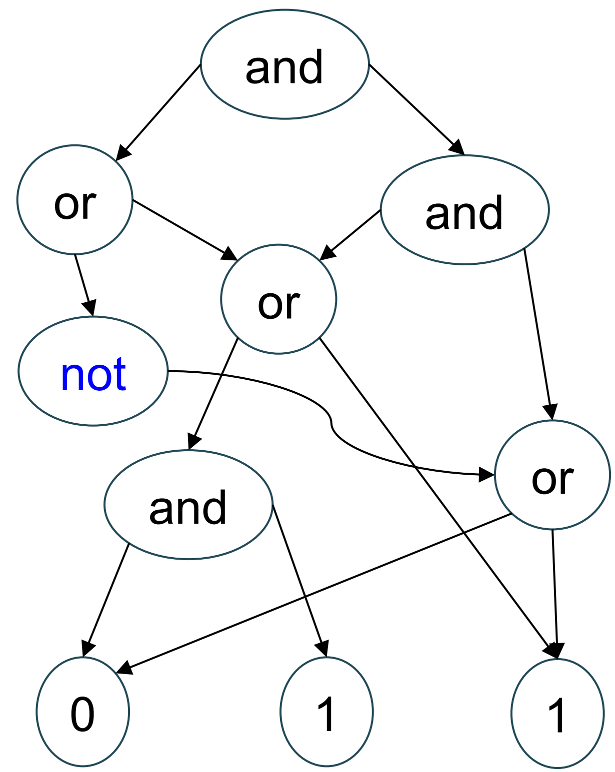


Example: Boolean Circuits

Find all gates whose value is 1

Schema: AND(u,x,y), OR(u,x,y), LEAF(u,0/1)
NOT(u,x)

Problem: we cannot use negation, or difference



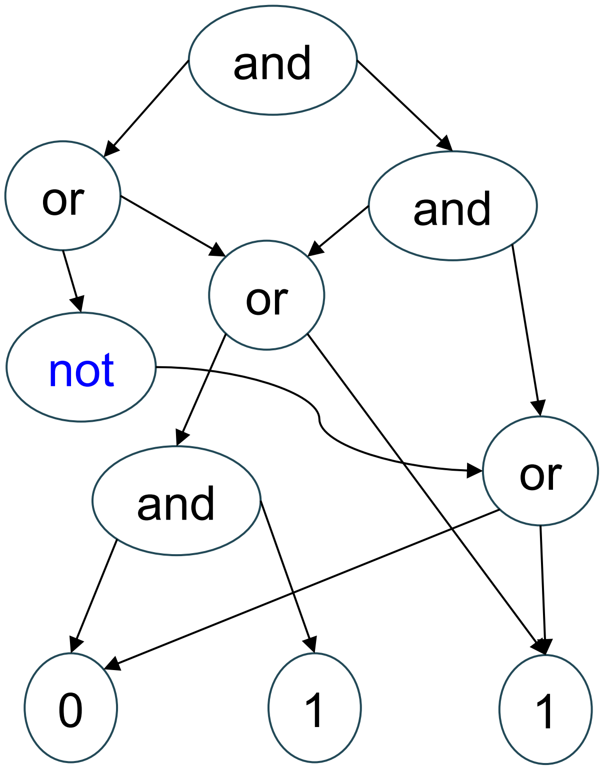
Solution: compute both the ONEs and the ZEROs

Example: Boolean Circuits

Find all gates whose value is 1

Schema: AND(u,x,y), OR(u,x,y), LEAF(u,0/1)
NOT(u,x)

ONE(u) :- LEAF(u,1)
ONE(u) :- AND(u,x,y),ONE(x),ONE(y)
ONE(u) :- OR(u,x,y),ONE(x)
ONE(u) :- OR(u,x,y),ONE(y)
ZERO(u) :- LEAF(u,0)

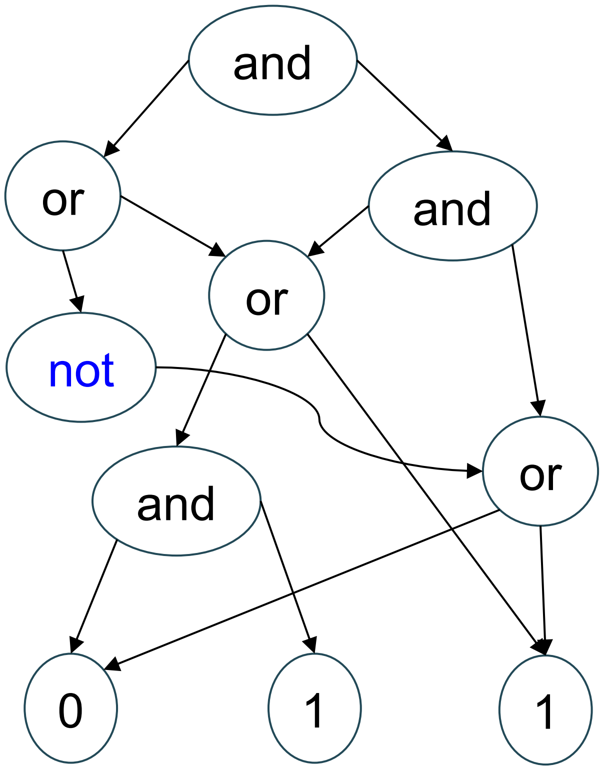


Example: Boolean Circuits

Find all gates whose value is 1

Schema: AND(u,x,y), OR(u,x,y), LEAF(u,0/1)
NOT(u,x)

```
ONE(u) :- LEAF(u,1)
ONE(u) :- AND(u,x,y),ONE(x),ONE(y)
ONE(u) :- OR(u,x,y),ONE(x)
ONE(u) :- OR(u,x,y),ONE(y)
ZERO(u) :- LEAF(u,0)
ZERO(u) :- AND(u,x,y),ZERO(x)
ZERO(u) :- AND(u,x,y),ZERO(y)
ZERO(u) :- OR(u,x,y),ZERO(x),ZERO(y)
```

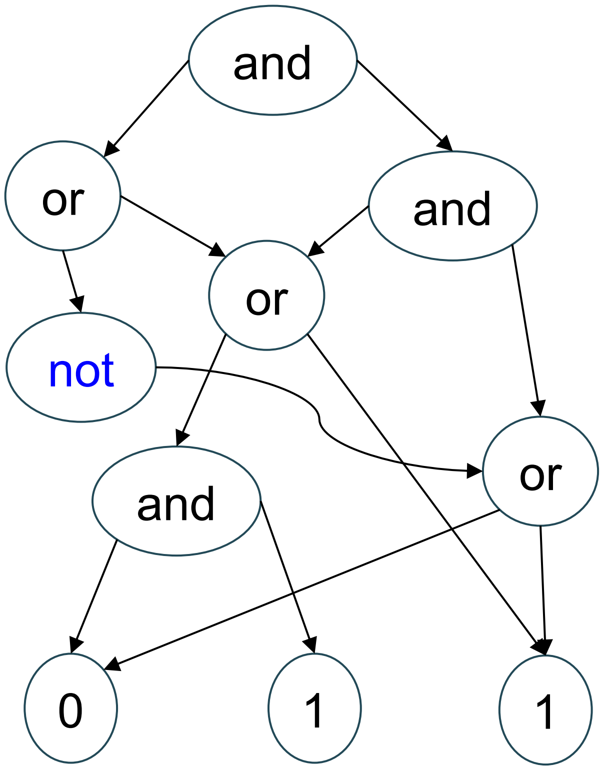


Example: Boolean Circuits

Find all gates whose value is 1

Schema: AND(u,x,y), OR(u,x,y), LEAF(u,0/1)
NOT(u,x)

```
ONE(u) :- LEAF(u,1)
ONE(u) :- AND(u,x,y),ONE(x),ONE(y)
ONE(u) :- OR(u,x,y),ONE(x)
ONE(u) :- OR(u,x,y),ONE(y)
ZERO(u) :- LEAF(u,0)
ZERO(u) :- AND(u,x,y),ZERO(x)
ZERO(u) :- AND(u,x,y),ZERO(y)
ZERO(u) :- OR(u,x,y),ZERO(x),ZERO(y)
ZERO(u) :- NOT(u,x), ONE(x)
ONE(u) :- NOT(u,x), ZERO(x)
```



Discussion

- Datalog can express some surprisingly complex queries
- Yet it is also very limited:
  - Cannot express set difference

$$C(x):-A(x),\neg B(x)$$

  - Cannot express arithmetic or aggregates:

$$C\left(x,sum(z)\right):-A(x,y),B(y,z)$$

- Will discuss next time how to extend it

Outline

- Examples
- Semi-naïve Evaluation
- Incremental View Maintenance

Naïve Evaluation Algorithm

Notations. Fix a datalog program P.

- Will denote the EDBs with I,
  and denote the IDBs with J
- Immediate consequence operator:
  maps I,J to new state J' = $T_P(J)$

Naïve Evaluation Algorithm

$J_0 = \emptyset$
for $t = 0, \infty$
  $J_{t+1} = T_P(J_t)$
  if $J_{t+1} = J_t$ break

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 Maintenace

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

```
V :- body
```

Incremental View Maintenace

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

V :- body

Some relations are updated: $R_1 \leftarrow R_1 \cup \Delta R_1$, $R_1 \leftarrow R_2 \cup \Delta R_2$, …

Incremental View Maintenace

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

V :- body

Some 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 updated: $V \leftarrow V \cup \Delta V$

Incremental View Maintenace

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

V :- body

Some 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 updated: $V \leftarrow V \cup \Delta V$

Incremental View Maintenance: Compute ΔV without computing V

Background: Incremental View Maintenace

Example 1:

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

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

Background: Incremental View Maintenace

Example 1:

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

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

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

Background: Incremental View Maintenace

Example 2:

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

If R ← R ∪ΔR and S ← S ∪ΔS then what is ΔV(x,y) ?

Background: Incremental View Maintenace

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 $\Delta V(x,y)$?

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

Background: Incremental View Maintenace

Example 3:

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

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

Background: Incremental View Maintenace

Example 3:

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

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

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

Naïve, Semi-naïve

Naïve:

$J_0 = \emptyset$
for $t = 0, \infty$
  $J_{t+1} = T_P(J_t)$
  if $J_{t+1} = J_t$ break

Naïve, Semi-naïve

Naïve:

$J_0 = \emptyset$
for $t = 0, \infty$
  $J_{t+1} = T_P(J_t)$
  if $J_{t+1} = J_t$ break

Semi-naïve:

$J_0 = \emptyset$
for $t = 0, \infty$
  $\Delta_t = T_P(J_t) - J_t$
  if $\Delta_t = \emptyset$ break
  $J_{t+1} = J_t \cup \Delta_t$

Naïve, Semi-naïve

Naïve:

$J_0 = \emptyset$
for $t = 0, \infty$
  $J_{t+1} = T_P(J_t)$
  if $J_{t+1} = J_t$ break

Semi-naïve:

$J_0 = \emptyset$
for $t = 0, \infty$
  $\Delta_t = T_P(J_t) - J_t$
  if $\Delta_t = \emptyset$ break
  $J_{t+1} = J_t \cup \Delta_t$

Semi-naïve with incremental computation

$J_0 = \emptyset,\ J_1 = \Delta_0 = T_P(\emptyset)$
for $t = 1, \infty$
  $\Delta_t = T_P(J_{t-1} \cup \Delta_{t-1}) - J_t$
  $= \Delta T_P(J_{t-1}, \Delta_{t-1}) - J_t$
  if $\Delta_t = \emptyset$ break
  $J_{t+1} = J_t \cup \Delta_t$

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

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

Example

$J_0 = \emptyset$
for $t = 0, \infty$
  $J_{t+1} = T_P(J_t)$
  if $J_{t+1} = J_t$ break

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

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

Example

$J_0 = \emptyset$
for t = 0, ∞
  $J_{t+1} = T_P(J_t)$
  if $J_{t+1} = J_t$ break

$T_0(x,y)$ = false

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

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

Example

$J_0 = \emptyset$
for $t = 0, \infty$
  $J_{t+1} = T_P(J_t)$
  if $J_{t+1} = J_t$ break

$T_0(x,y) = \text{false}$

for $t = 0, \infty$

$$T_{t+1}(x,y) = R(x,y) \lor (R(x,z) \land T_t(z,y))$$

$T(x,y) \text{ :- } R(x,y)$

$T(x,y) \text{ :- } R(x,z), T(z,y)$

Example

$J_0 = \emptyset$
for $t = 0, \infty$
  $J_{t+1} = T_P(J_t)$
  if $J_{t+1} = J_t$ break

$T_0(x,y) = \text{false}$

for $t = 0, \infty$

  $T_{t+1}(x,y) = R(x,y) \vee (R(x,z) \wedge T_t(z,y))$

  if $T_{t+1} = T_t$ break

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

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

Example

$J_0 = \emptyset$
for $t = 0, \infty$
  $J_{t+1} = T_P(J_t)$
  if $J_{t+1} = J_t$ break

$J_0 = \emptyset$, $J_1 = \Delta_0 = T_P(\emptyset)$
for $t = 1, \infty$
  $\Delta_t = \Delta T_P(J_{t-1}, \Delta_{t-1}) - J_t$
  if $\Delta_t = \emptyset$ break
  $J_{t+1} = J_t \cup \Delta_t$

$T_0(x,y) = \text{false}$
for $t = 0, \infty$
  $T_{t+1}(x,y) = R(x,y)$
    $\vee (R(x,z) \wedge T_t(z,y))$
  if $T_{t+1} = T_t$ break

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

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

Example

$J_0 = \emptyset$
for $t = 0, \infty$
  $J_{t+1} = T_P(J_t)$
  if $J_{t+1} = J_t$ break

$J_0 = \emptyset$, $J_1 = \Delta_0 = T_P(\emptyset)$
for $t = 1, \infty$
  $\Delta_t = \Delta T_P(J_{t-1}, \Delta_{t-1}) - J_t$
  if $\Delta_t = \emptyset$ break
  $J_{t+1} = J_t \cup \Delta_t$

$T_0(x,y)$ = false
for $t = 0, \infty$
  $T_{t+1}(x,y) = R(x,y) \vee (R(x,z) \wedge T_t(z,y))$
  if $T_{t+1} = T_t$ break

$T_0(x,y)$ = false, $T_1(x,y) = \Delta_0(x,y) = R(x,y)$

$T(x,y) \text{ :- } R(x,y)$

$T(x,y) \text{ :- } R(x,z), T(z,y)$

Example

$J_0 = \emptyset$
for $t = 0, \infty$
  $J_{t+1} = T_P(J_t)$
  if $J_{t+1} = J_t$ break

$J_0 = \emptyset$, $J_1 = \Delta_0 = T_P(\emptyset)$
for $t = 1, \infty$
  $\Delta_t = \Delta T_P(J_{t-1}, \Delta_{t-1}) - J_t$
  if $\Delta_t = \emptyset$ break
  $J_{t+1} = J_t \cup \Delta_t$

$T_0(x,y) = \text{false}$
for $t = 0, \infty$
  $T_{t+1}(x,y) = R(x,y) \vee (R(x,z) \wedge T_t(z,y))$
  if $T_{t+1} = T_t$ break

$T_0(x,y) = \text{false}$, $T_1(x,y) = \Delta_0(x,y) = R(x,y)$
for $t = 1, \infty$
  $\Delta_t(x,y) = (R(x,z) \wedge \Delta_{t-1}(z,y)) \wedge \neg T_t(x,y)$

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

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

Example

$J_0 = \emptyset$
for $t = 0, \infty$
  $J_{t+1} = T_P(J_t)$
  if $J_{t+1} = J_t$ break

$J_0 = \emptyset$, $J_1 = \Delta_0 = T_P(\emptyset)$
for $t = 1, \infty$
  $\Delta_t = \Delta T_P(J_{t-1}, \Delta_{t-1}) - J_t$
  if $\Delta_t = \emptyset$ break
  $J_{t+1} = J_t \cup \Delta_t$

$T_0(x,y)$ = false
for $t = 0, \infty$
  $T_{t+1}(x,y) = R(x,y)$
    $\vee (R(x,z) \wedge T_t(z,y))$
  if $T_{t+1} = T_t$ break

$T_0(x,y)$ = false, $T_1(x,y) = \Delta_0(x,y) = R(x,y)$
for $t = 1, \infty$
  $\Delta_t(x,y) = (R(x,z) \wedge \Delta_{t-1}(z,y)) \wedge \neg T_t(x,y)$
  if $\Delta_t = \emptyset$ break

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

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

Example

$J_0 = \emptyset$
for $t = 0, \infty$
  $J_{t+1} = T_P(J_t)$
  if $J_{t+1} = J_t$ break

$J_0 = \emptyset$, $J_1 = \Delta_0 = T_P(\emptyset)$
for $t = 1, \infty$
  $\Delta_t = \Delta T_P(J_{t-1}, \Delta_{t-1}) - J_t$
  if $\Delta_t = \emptyset$ break
  $J_{t+1} = J_t \cup \Delta_t$

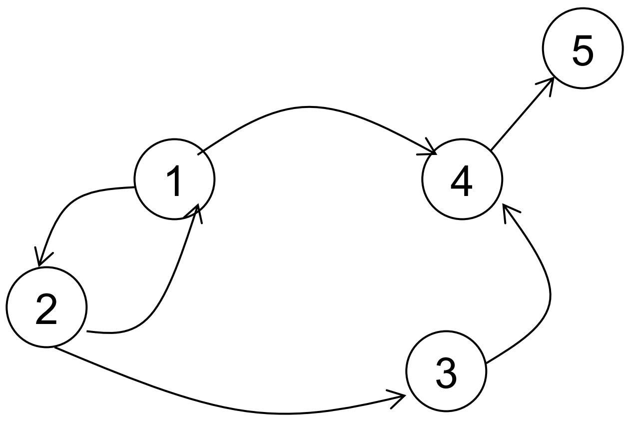
$T_0(x,y) = \text{false}$
for $t = 0, \infty$
  $T_{t+1}(x,y) = R(x,y) \vee (R(x,z) \wedge T_t(z,y))$
  if $T_{t+1} = T_t$ break

$T_0(x,y) = \text{false}$, $T_1(x,y) = \Delta_0(x,y) = R(x,y)$
for $t = 1, \infty$
  $\Delta_t(x,y) = (R(x,z) \wedge \Delta_{t-1}(z,y)) \wedge \neg T_t(x,y)$
  if $\Delta_t = \emptyset$ break
  $T_{t+1}(x,y) = T_t(x,y) \vee \Delta_t(x,y)$

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

Example

$T_0(x,y)$ = false, $T_1(x,y) = \Delta_0(x,y) = R(x,y)$
for t = 1, ∞
  $\Delta_t(x,y) = (R(x,z) \wedge \Delta_{t-1}(z,y)) \wedge \neg T_t(x,y)$
  if $\Delta_t = \emptyset$ break
  $T_{t+1}(x,y) = T_t(x,y) \vee \Delta_t(x,y)$



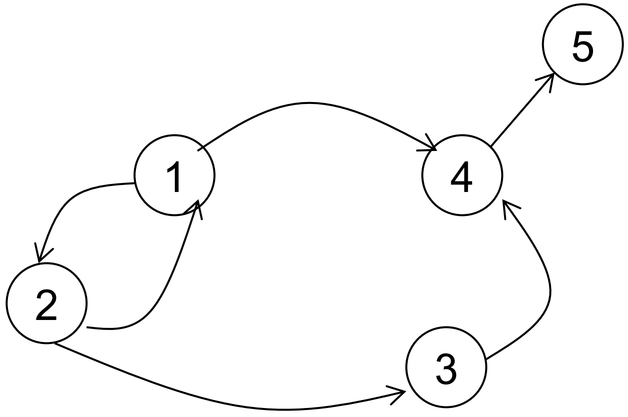
R=

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

Example

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

$T_0(x,y)$ = false, $T_1(x,y) = \Delta_0(x,y) = R(x,y)$
for t = 1, ∞
  $\Delta_t(x,y) = (R(x,z) \wedge \Delta_{t-1}(z,y)) \wedge \neg T_t(x,y)$
  if $\Delta_t = \emptyset$ break
  $T_{t+1}(x,y) = T_t(x,y) \vee \Delta_t(x,y)$



R=

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

Δ_0

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

T_1=

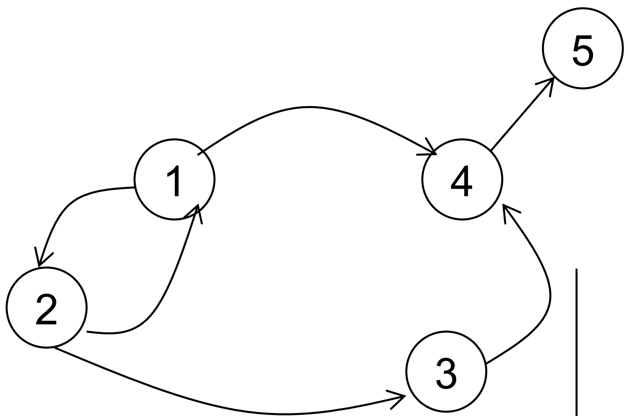
1	2
1	4
2	1
2	3
3	4
4	5

t=0

Example

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

$T_0(x,y)$ = false, $T_1(x,y) = \Delta_0(x,y) = R(x,y)$
for t = 1, ∞
  $\Delta_t(x,y) = (R(x,z) \wedge \Delta_{t-1}(z,y)) \wedge \neg T_t(x,y)$
  if $\Delta_t = \emptyset$ break
  $T_{t+1}(x,y) = T_t(x,y) \vee \Delta_t(x,y)$



t=0

R=

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

Δ_0

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

T_1=

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

t=1

Δ_1 = paths of length 2

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

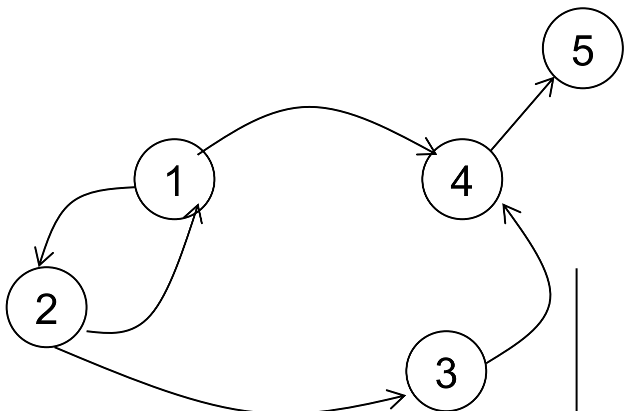
T_2 = path of length ≤ 2

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

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

$T_0(x,y)$ = false, $T_1(x,y) = \Delta_0(x,y) = R(x,y)$
for t = 1, ∞
  $\Delta_t(x,y) = (R(x,z) \wedge \Delta_{t-1}(z,y)) \wedge \neg T_t(x,y)$
  if $\Delta_t = \emptyset$ break
  $T_{t+1}(x,y) = T_t(x,y) \vee \Delta_t(x,y)$

Example



t=0

R=

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

Δ_0

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

T_1=

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

t=1

Δ_1 = paths of length 2

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

T_2 = path of length ≤ 2

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

t=2

Δ_2 = paths of length 3

2	5

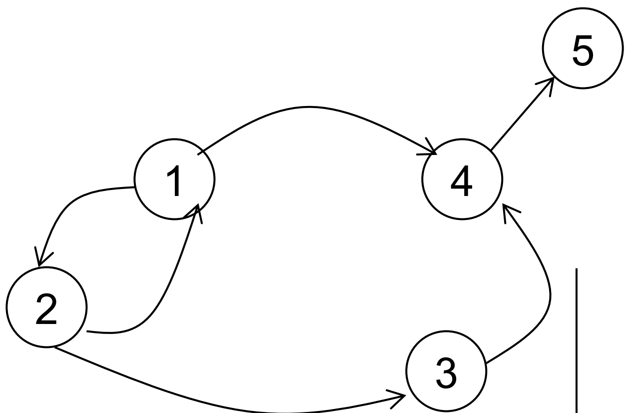
T_3= path of length ≤ 3

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

Example

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

$T_0(x,y)$ = false, $T_1(x,y) = \Delta_0(x,y) = R(x,y)$
for t = 1, ∞
  $\Delta_t(x,y) = (R(x,z) \wedge \Delta_{t-1}(z,y)) \wedge \neg T_t(x,y)$
  if $\Delta_t = \emptyset$ break
  $T_{t+1}(x,y) = T_t(x,y) \vee \Delta_t(x,y)$



t=0

R=

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

Δ_0

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

T_1=

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

t=1

Δ_1 = paths of length 2

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

T_2 = path of length ≤ 2

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

t=2

Δ_2 = paths of length 3

2	5

T_3= path of length ≤ 3

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

t=3

Δ_3 = paths of length 4

Discussion

- 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 leads to one incremental rule
  - A non-linear rule leads to many incremental rules -- next

Non-linear Delta-Rules

…

$\Delta_t = \Delta T_P(J_{t-1}, \Delta_{t-1}) - J_t$

$J_{t+1} = J_t \cup \Delta_t$

T :- A,B,C

$(\Delta T)_t$:- $(\Delta A)_{t-1}, B_{t-1}, C_{t-1}$

Non-linear Delta-Rules

…

$\Delta_t = \Delta T_P(J_{t-1}, \Delta_{t-1}) - J_t$

$J_{t+1} = J_t \cup \Delta_t$

T :- A,B,C

$(\Delta T)_t$:- $(\Delta A)_{t-1}, B_{t-1}, C_{t-1}$

$(\Delta T)_t$:- $A_{t-1}, (\Delta B)_{t-1}, C_{t-1}$

Non-linear Delta-Rules

…

$\Delta_t = \Delta T_P(J_{t-1}, \Delta_{t-1}) - J_t$

$J_{t+1} = J_t \cup \Delta_t$

T :- A,B,C

$(\Delta T)_t$:- $(\Delta A)_{t-1}, B_{t-1}, C_{t-1}$

$(\Delta T)_t$:- $A_{t-1}, (\Delta B)_{t-1}, C_{t-1}$

$(\Delta T)_t$:- $A_{t-1}, B_{t-1}, (\Delta C)_{t-1}$

Non-linear Delta-Rules

…

$\Delta_t = \Delta T_P(J_{t-1}, \Delta_{t-1}) - J_t$

$J_{t+1} = J_t \cup \Delta_t$

T :- A,B,C

$(\Delta T)_t$:- $(\Delta A)_{t-1}, B_{t-1}, C_{t-1}$

$(\Delta T)_t$:- $A_{t-1}, (\Delta B)_{t-1}, C_{t-1}$

$(\Delta T)_t$:- $A_{t-1}, B_{t-1}, (\Delta C)_{t-1}$

$(\Delta T)_t$:- $(\Delta A)_{t-1}$, $(\Delta B)_{t-1}, C_{t-1}$

Non-linear Delta-Rules

…

$\Delta_t = \Delta T_P(J_{t-1}, \Delta_{t-1}) - J_t$

$J_{t+1} = J_t \cup \Delta_t$

T :- A,B,C

$(\Delta T)_t$:- $(\Delta A)_{t-1}, B_{t-1}, C_{t-1}$

$(\Delta T)_t$:- $A_{t-1}, (\Delta B)_{t-1}, C_{t-1}$

$(\Delta T)_t$:- $A_{t-1}, B_{t-1}, (\Delta C)_{t-1}$

$(\Delta T)_t$:- $(\Delta A)_{t-1}, (\Delta B)_{t-1}, C_{t-1}$

$(\Delta T)_t$:- $(\Delta A)_{t-1}, B_{t-1}, (\Delta C)_{t-1}$

$(\Delta T)_t$:- $A_{t-1}, (\Delta B)_{t-1}, (\Delta C)_{t-1}$

$(\Delta T)_t$:- $(\Delta A)_{t-1}, (\Delta B)_{t-1}, (\Delta C)_{t-1}$

Non-linear Delta-Rules

…

$\Delta_t = \Delta T_P(J_{t-1}, \Delta_{t-1}) - J_t$

$J_{t+1} = J_t \cup \Delta_t$

Also add $\neg T_t$ to each rule

T :- A,B,C

$(\Delta T)_t$:- $(\Delta A)_{t-1}, B_{t-1}, C_{t-1}$

$(\Delta T)_t$:- $A_{t-1}, (\Delta B)_{t-1}, C_{t-1}$

$(\Delta T)_t$:- $A_{t-1}, B_{t-1}, (\Delta C)_{t-1}$

$(\Delta T)_t$:- $(\Delta A)_{t-1}, (\Delta B)_{t-1}, C_{t-1}$

$(\Delta T)_t$:- $(\Delta A)_{t-1}, B_{t-1}, (\Delta C)_{t-1}$

$(\Delta T)_t$:- $A_{t-1}, (\Delta B)_{t-1}, (\Delta C)_{t-1}$

$(\Delta T)_t$:- $(\Delta A)_{t-1}, (\Delta B)_{t-1}, (\Delta C)_{t-1}$

Non-linear Delta-Rules

…

$\Delta_t = \Delta T_P(J_{t-1}, \Delta_{t-1}) - J_t$

$J_{t+1} = J_t \cup \Delta_t$

Also add $\neg T_t$ to each rule

T :- A,B,C

$(\Delta T)_t$:- $(\Delta A)_{t-1}, B_{t-1}, C_{t-1}$

$(\Delta T)_t$:- $A_{t-1}, (\Delta B)_{t-1}, C_{t-1}$

$(\Delta T)_t$:- $A_{t-1}, B_{t-1}, (\Delta C)_{t-1}$

$(\Delta T)_t$:- $(\Delta A)_{t-1}, (\Delta B)_{t-1}, C_{t-1}$

$(\Delta T)_t$:- $(\Delta A)_{t-1}, B_{t-1}, (\Delta C)_{t-1}$

$(\Delta T)_t$:- $A_{t-1}, (\Delta B)_{t-1}, (\Delta C)_{t-1}$

$(\Delta T)_t$:- $(\Delta A)_{t-1}, (\Delta B)_{t-1}, (\Delta C)_{t-1}$

2^n-1 rules

Non-linear Delta-Rules

…

$\Delta_t = \Delta T_P(J_{t-1}, \Delta_{t-1}) - J_t$

$J_{t+1} = J_t \cup \Delta_t$

Also add $\neg T_t$ to each rule

T :- A,B,C

$(\Delta T)_t$:- $(\Delta A)_{t-1},B_{t-1},C_{t-1}$
$(\Delta T)_t$:- $A_{t-1},(\Delta B)_{t-1},C_{t-1}$
$(\Delta T)_t$:- $A_{t-1},B_{t-1},(\Delta C)_{t-1}$
$(\Delta T)_t$:- $(\Delta A)_{t-1}, (\Delta B)_{t-1},C_{t-1}$
$(\Delta T)_t$:- $(\Delta A)_{t-1},B_{t-1},(\Delta C)_{t-1}$
$(\Delta T)_t$:- $A_{t-1},(\Delta B)_{t-1},(\Delta C)_{t-1}$
$(\Delta T)_t$:- $(\Delta A)_{t-1},(\Delta B)_{t-1},(\Delta C)_{t-1}$

2^n-1 rules

$(\Delta T)_t$:- $(\Delta A)_{t-1},B_{t-1},C_{t-1}$
$(\Delta T)_t$:- $A_t,(\Delta B)_{t-1},C_{t-1}$

Non-linear Delta-Rules

…

$\Delta_t = \Delta T_P(J_{t-1}, \Delta_{t-1}) - J_t$

$J_{t+1} = J_t \cup \Delta_t$

Also add $\neg T_t$ to each rule

T :- A,B,C

$(\Delta T)_t$:- $(\Delta A)_{t-1}, B_{t-1}, C_{t-1}$
$(\Delta T)_t$:- $A_{t-1}, (\Delta B)_{t-1}, C_{t-1}$
$(\Delta T)_t$:- $A_{t-1}, B_{t-1}, (\Delta C)_{t-1}$
$(\Delta T)_t$:- $(\Delta A)_{t-1}, (\Delta B)_{t-1}, C_{t-1}$
$(\Delta T)_t$:- $(\Delta A)_{t-1}, B_{t-1}, (\Delta C)_{t-1}$
$(\Delta T)_t$:- $A_{t-1}, (\Delta B)_{t-1}, (\Delta C)_{t-1}$
$(\Delta T)_t$:- $(\Delta A)_{t-1}, (\Delta B)_{t-1}, (\Delta C)_{t-1}$

2^n-1 rules

$(\Delta T)_t$:- $(\Delta A)_{t-1}, B_{t-1}, C_{t-1}$
$(\Delta T)_t$:- $A_t, (\Delta B)_{t-1}, C_{t-1}$
$(\Delta T)_t$:- $A_t, B_t, (\Delta C)_{t-1}$

Non-linear Delta-Rules

…

$\Delta_t = \Delta T_P(J_{t-1}, \Delta_{t-1}) - J_t$

$J_{t+1} = J_t \cup \Delta_t$

Also add $\neg T_t$ to each rule

T :- A,B,C

$(\Delta T)_t$:- $(\Delta A)_{t-1}, B_{t-1}, C_{t-1}$
$(\Delta T)_t$:- $A_{t-1}, (\Delta B)_{t-1}, C_{t-1}$
$(\Delta T)_t$:- $A_{t-1}, B_{t-1}, (\Delta C)_{t-1}$
$(\Delta T)_t$:- $(\Delta A)_{t-1}, (\Delta B)_{t-1}, C_{t-1}$
$(\Delta T)_t$:- $(\Delta A)_{t-1}, B_{t-1}, (\Delta C)_{t-1}$
$(\Delta T)_t$:- $A_{t-1}, (\Delta B)_{t-1}, (\Delta C)_{t-1}$
$(\Delta T)_t$:- $(\Delta A)_{t-1}, (\Delta B)_{t-1}, (\Delta C)_{t-1}$

2^n-1 rules

$(\Delta T)_t$:- $(\Delta A)_{t-1}, B_{t-1}, C_{t-1}$
$(\Delta T)_t$:- $A_t, (\Delta B)_{t-1}, C_{t-1}$
$(\Delta T)_t$:- $A_t, B_t, (\Delta C)_{t-1}$

Only n rules, but IDB at time t is >> IDB at t-1

Discussion

- All datalog engines support the semi-naïve algorithm
- Some support a more advanced optimization called magic sets
- Semi-naïve is a simple special case of Incremental View Maintenance (IVM)

Let's see IVM for aggregates next

Outline

- Examples
- Semi-naïve Evaluation
- Incremental View Maintenance

IVM

- We have materialized some view
- There is small update to the input
- IVM computes the update to the view
- Best done by converting queries to tensor kernels

Example

Customer(cno, cname, ccity)

Order(cno, pno, wno, quant)

Product(pno, pname, pprice)

Warehouse(wno, wname, wstate)

For each city, compute
the revenue for all orders
to that city from the state 'WA'

Example

Customer(cno, cname, ccity)

Order(cno, pno, wno, quant)

Product(pno, pname, pprice)

Warehouse(wno, wname, wstate)

For each city, compute the revenue for all orders to that city from the state 'WA'

```
SELECT C.ccity, sum(O.quant * P.pprice)
FROM C, O, P, W
WHERE C.cno = O.cno and O.pno = P.pno
    and O.wno = W.wno and W.wstate = 'WA'
GROUP BY C.ccity
```

Example

Customer(cno, cname, ccity)
Order(cno, pno, wno, quant)
Product(pno, pname, pprice)
Warehouse(wno, wname, wstate)

For each city, compute the revenue for all orders to that city from the state 'WA'

```
SELECT C.ccity, sum(O.quant * P.pprice)
FROM C, O, P, W
WHERE C.cno = O.cno and O.pno = P.pno
    and O.wno = W.wno and W.wstate = 'WA'
GROUP BY C.ccity
```

$$\text{Order} = \text{Order} \cup \Delta\text{Order}$$

K-Relations

- A standard relation maps tuples to {0,1}
  - Product(p032, ‘iPhone’, 499) = 1
  - Product(p032, ‘iPad’, 499) = 0
  - Customer(c55, ‘Alice’, ‘Portland’) = 1

K-Relations

- A standard relation maps tuples to {0,1}
  - Product(p032, ‘iPhone’, 499) = 1
  - Product(p032, ‘iPad’, 499) = 0
  - Customer(c55, ‘Alice’, ‘Portland’) = 1
- A K-relation maps tuples to semiring K (we will only use the semiring of reals)

K-Relations

- A standard relation maps tuples to {0,1}
  - Product(p032, ‘iPhone’, 499) = 1
  - Product(p032, ‘iPad’, 499) = 0
  - Customer(c55, ‘Alice’, ‘Portland’) = 1
- A K-relation maps tuples to semiring K (we will only use the semiring of reals)
  - P[p032, ‘iPhone’] = 499

K-Relations

- A standard relation maps tuples to {0,1}
  - Product(p032, ‘iPhone’, 499) = 1
  - Product(p032, ‘iPad’, 499) = 0
  - Customer(c55, ‘Alice’, ‘Portland’) = 1
- A K-relation maps tuples to semiring K (we will only use the semiring of reals)
  - P[p032, ‘iPhone’] = 499
  - C[c55, ‘Alice’, ‘Portland’] = 1

Customer(cno, cname, ccity)
Order(cno, pno, wno, quant)
Product(pno, pname, pprice)
Warehouse(wno, wname, wstate)

Example

C[cno, cname, ccity] = 0 or 1
O[cno, pno, wno] = quant
P[pno, pname] = pprice
W[wno, wname, wstate] = 0 or 1

Example

Customer(cno, cname, ccity)
Order(cno, pno, wno, quant)
Product(pno, pname, pprice)
Warehouse(wno, wname, wstate)

C[cno, cname, ccity] = 0 or 1
O[cno, pno, wno] = quant
P[pno, pname] = pprice
W[wno, wname, wstate] = 0 or 1

```
SELECT C.ccity, sum(O.quant * P.pprice)
FROM C, O, P, W
WHERE C.cno = O.cno and O.pno = P.pno
    and O.wno = W.wno and W.wstate = 'WA'
GROUP BY C.ccity
```

Customer(cno, cname, ccity)
Order(cno, pno, wno, quant)
Product(pno, pname, pprice)
Warehouse(wno, wname, wstate)

Example

C[cno, cname, ccity] = 0 or 1
O[cno, pno, wno] = quant
P[pno, pname] = pprice
W[wno, wname, wstate] = 0 or 1

```
SELECT C.ccity, sum(O.quant * P.pprice)
FROM C, O, P, W
WHERE C.cno = O.cno and O.pno = P.pno
    and O.wno = W.wno and W.wstate = 'WA'
GROUP BY C.ccity
```

$$\mathrm{Answ}[\mathrm{ccity}] = \sum_{\substack{\mathrm{cno}, \\ \mathrm{pno}, \\ \mathrm{wno}, \\ \mathrm{cname}, \\ \ldots}} \mathrm{C}[\mathrm{cno}, \mathrm{cname}, \mathrm{ccity}] * \mathrm{O}[\mathrm{cno}, \mathrm{pno}, \mathrm{wno}] * \mathrm{P}[\mathrm{pno}, \mathrm{pname}] * \mathrm{W}[\mathrm{wno}, \mathrm{wname}, '\mathrm{WA}']$$

Example

Customer(cno, cname, ccity)
Order(cno, pno, wno, quant)
Product(pno, pname, pprice)
Warehouse(wno, wname, wstate)

C[cno, cname, ccity] = 0 or 1
O[cno, pno, wno] = quant
P[pno, pname] = pprice
W[wno, wname, wstate] = 0 or 1

```
SELECT C.ccity, sum(O.quant * P.pprice)
FROM C, O, P, W
WHERE C.cno = O.cno and O.pno = P.pno
    and O.wno = W.wno and W.wstate = 'WA'
GROUP BY C.ccity
```

Query = tensor kernel

$$\text{Answ}[\text{ccity}] = \sum_{\substack{\text{cno},\\ \text{pno},\\ \text{wno},\\ \text{cname},\\ \dots}} \text{C}[\text{cno}, \text{cname}, \text{ccity}] * \text{O}[\text{cno}, \text{pno}, \text{wno}] * \text{P}[\text{pno}, \text{pname}] * \text{W}[\text{wno}, \text{wname}, '\text{WA}']$$

Customer(cno, cname, ccity)
Order(cno, pno, wno, quant)
Product(pno, pname, pprice)
Warehouse(wno, wname, wstate)

Example

C[cno, cname, ccity] = 0 or 1
O[cno, pno, wno] = quant
P[pno, pname] = pprice
W[wno, wname, wstate] = 0 or 1

```
SELECT C.ccity, sum(O.quant * P.pprice)
FROM C, O, P, W
WHERE C.cno = O.cno and O.pno = P.pno
    and O.wno = W.wno and W.wstate = 'WA'
GROUP BY C.ccity
```

Query = tensor kernel

$$\text{Answ}[\text{ccity}] = \sum_{\substack{\text{cno,}\\\text{pno,}\\\text{wno,}\\\text{cname,}\\\ldots}} \text{C}[\text{cno}, \text{cname}, \text{ccity}] * \text{O}[\text{cno}, \text{pno}, \text{wno}] * \text{P}[\text{pno}, \text{pname}] * \text{W}[\text{wno}, \text{wname}, '\text{WA}']$$

Differentiate tensor kernel

$$\Delta\text{Answ}[\text{ccity}] = \sum_{\substack{\text{cno,}\\\text{pno,}\\\text{wno,}\\\text{cname,}\\\ldots}} \text{C}[\text{cno}, \text{cname}, \text{ccity}] * \Delta\text{O}[\text{cno}, \text{pno}, \text{wno}] * \text{P}[\text{pno}, \text{pname}] * \text{W}[\text{wno}, \text{wname}, '\text{WA}']$$

Customer(cno, cname, ccity)
Order(cno, pno, wno, quant)
Product(pno, pname, pprice)
Warehouse(wno, wname, wstate)

Example

C[cno, cname, ccity] = 0 or 1
O[cno, pno, wno] = quant
P[pno, pname] = pprice
W[wno, wname, wstate] = 0 or 1

```
SELECT C.ccity, sum(O.quant * P.pprice)
FROM C, O, P, W
WHERE C.cno = O.cno and O.pno = P.pno
    and O.wno = W.wno and W.wstate = 'WA'
GROUP BY C.ccity
```

Insert into Order values (c55,p032,w99,5)

$$\mathrm{Answ}[\mathrm{ccity}] = \sum_{\substack{\mathrm{cno},\\ \mathrm{pno},\\ \mathrm{wno},\\ \mathrm{cname},\\ \ldots}} C[\mathrm{cno}, \mathrm{cname}, \mathrm{ccity}] * O[\mathrm{cno}, \mathrm{pno}, \mathrm{wno}] * P[\mathrm{pno}, \mathrm{pname}] * W[\mathrm{wno}, \mathrm{wname}, '\mathrm{WA}']$$

$$\Delta\mathrm{Answ}[\mathrm{ccity}] = \sum_{\substack{\mathrm{cno},\\ \mathrm{pno},\\ \mathrm{wno},\\ \mathrm{cname},\\ \ldots}} C[\mathrm{cno}, \mathrm{cname}, \mathrm{ccity}] * \Delta O[\mathrm{cno}, \mathrm{pno}, \mathrm{wno}] * P[\mathrm{pno}, \mathrm{pname}] * W[\mathrm{wno}, \mathrm{wname}, '\mathrm{WA}']$$

Example

Customer(cno, cname, ccity)
Order(cno, pno, wno, quant)
Product(pno, pname, pprice)
Warehouse(wno, wname, wstate)

C[cno, cname, ccity] = 0 or 1
O[cno, pno, wno] = quant
P[pno, pname] = pprice
W[wno, wname, wstate] = 0 or 1

```
SELECT C.ccity, sum(O.quant * P.pprice)
FROM C, O, P, W
WHERE C.cno = O.cno and O.pno = P.pno
    and O.wno = W.wno and W.wstate = 'WA'
GROUP BY C.ccity
```

Insert into Order values (c55,p032,w99,5)

```
SELECT C.ccity, sum(O.quant * P.pprice)
FROM C, P, W
WHERE C.cno = 'c55' and 'p032' = P.pno
    and 'w99' = W.wno and W.wstate = 'WA'
```

$$\text{Answ}[\text{ccity}] = \sum_{\substack{\text{cno},\\ \text{pno},\\ \text{wno},\\ \text{cname},\\ \ldots}} \text{C}[\text{cno}, \text{cname}, \text{ccity}] * \text{O}[\text{cno}, \text{pno}, \text{wno}] * \text{P}[\text{pno}, \text{pname}] * \text{W}[\text{wno}, \text{wname}, '\text{WA}']$$

$$\Delta\text{Answ}[\text{ccity}] = \sum_{\substack{\text{cno},\\ \text{pno},\\ \text{wno},\\ \text{cname},\\ \ldots}} \text{C}[\text{cno}, \text{cname}, \text{ccity}] * \Delta\text{O}[\text{cno}, \text{pno}, \text{wno}] * \text{P}[\text{pno}, \text{pname}] * \text{W}[\text{wno}, \text{wname}, '\text{WA}']$$

Rings and Semirings

- A semiring (K,+,*,0,1) is ”a ring without difference”
  - $B = (\{0,1\}, \vee, \wedge, 0,1)$ is a semiring

Rings and Semirings

- A semiring (K,+,*,0,1) is ”a ring without difference”
  - $B = (\{0,1\}, \vee, \wedge, 0,1)$ is a semiring
  - $(N, +, *, 0,1)$ is a semiring

Rings and Semirings

- A semiring (K,+,*,0,1) is ”a ring without difference”
  - $B = (\{0,1\}, \vee, \wedge, 0,1)$ is a semiring
  - $(N, +, *, 0,1)$ is a semiring
  - $(R, +, *, 0,1)$ is a ring ($x - y$ exists)

Rings and Semirings

- A semiring (K,+,*,0,1) is ”a ring without difference”
  - $B = (\{0,1\}, \vee, \wedge, 0,1)$ is a semiring
  - $(N, +, *, 0,1)$ is a semiring
  - $(R, +, *, 0,1)$ is a ring ($x - y$ exists)
- Queries with set semantics use B

Rings and Semirings

- A semiring (K,+,*,0,1) is ”a ring without difference”
  - $B = (\{0,1\}, \vee, \wedge, 0,1)$ is a semiring
  - $(N, +, *, 0,1)$ is a semiring
  - $(R, +, *, 0,1)$ is a ring ($x - y$ exists)
- Queries with set semantics use B
- Queries with bag semantics use N

Rings and Semirings

- A semiring (K,+,*,0,1) is ”a ring without difference”
  - $B = (\{0,1\}, \vee, \wedge, 0,1)$ is a semiring
  - $(N, +, *, 0,1)$ is a semiring
  - $(R, +, *, 0,1)$ is a ring ($x - y$ exists)
- Queries with set semantics use B
- Queries with bag semantics use N
- Queries with aggregates use R (tensors)

Other Semirings

- Tropical semiring:

$$Trop = ([0, \infty], min, +, \infty, 0)$$

- Viterbi semiring:

$$V = ([0,1], max, *, 0,1)$$

- Any distributive lattice
- …

Next Lecture

- Extensions of datalog with non-monotone operators
- Needed for HW4