CSE544: Principles of Database Systems

Datalog

Announcements

- Review 7 (Datalog) due next Monday
- Project Milestone due next Friday
- HW3 is posted, due the following Monday

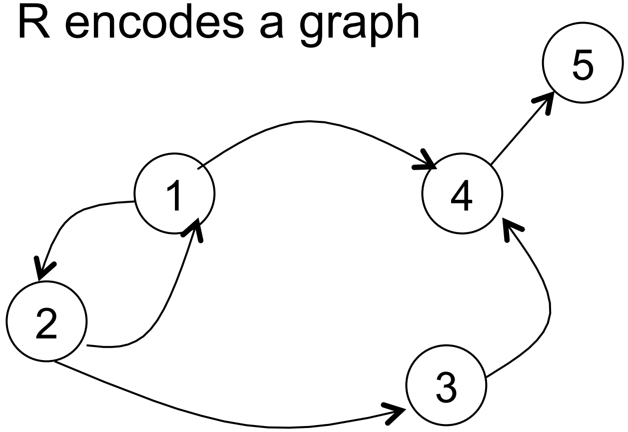
Datalog

Review the following basic concepts from Lecture 5:

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

Simple datalog programs

R encodes a graph



R=

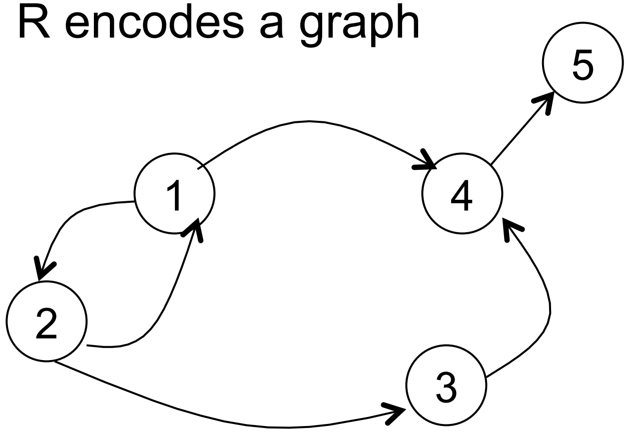
1	2
2	1
2	3
1	4
3	4
4	5

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?

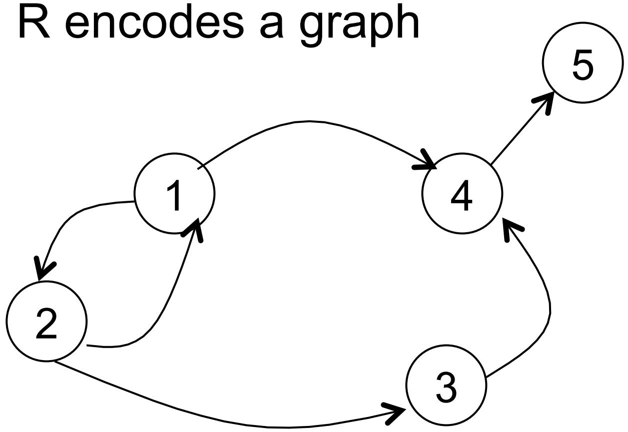
R=

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

Initially:
T is empty.

Simple datalog programs

R encodes a graph



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

What does it compute?

R=

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

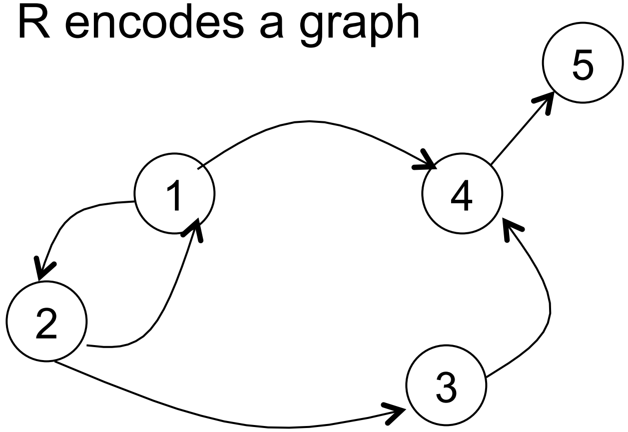
Initially:
T is empty.

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 =

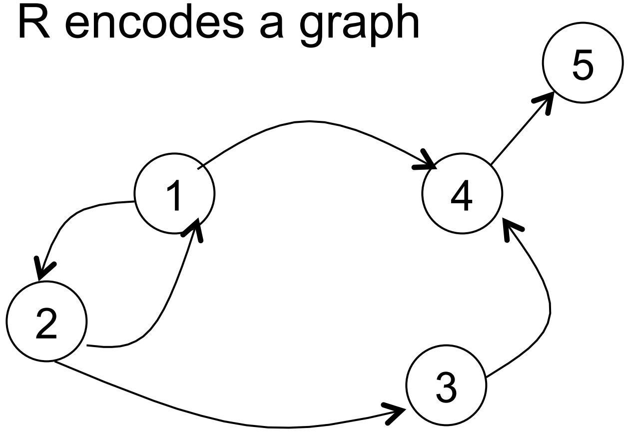
1	2
2	1
2	3
1	4
3	4
4	5

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

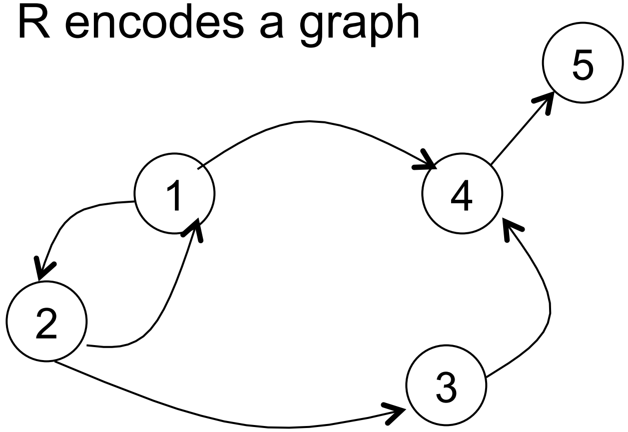
Third iteration:
T =

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

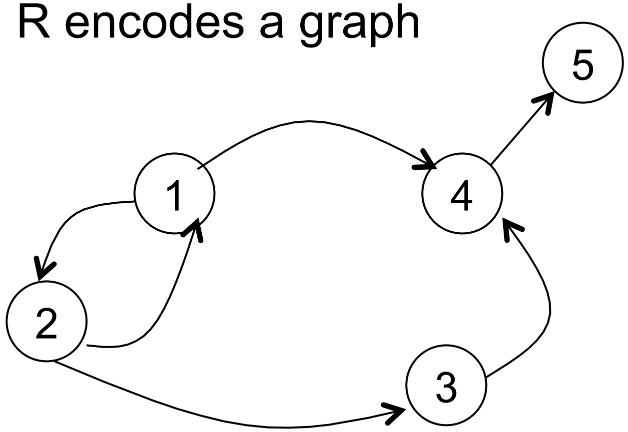
Discovered 3 times!

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)
```

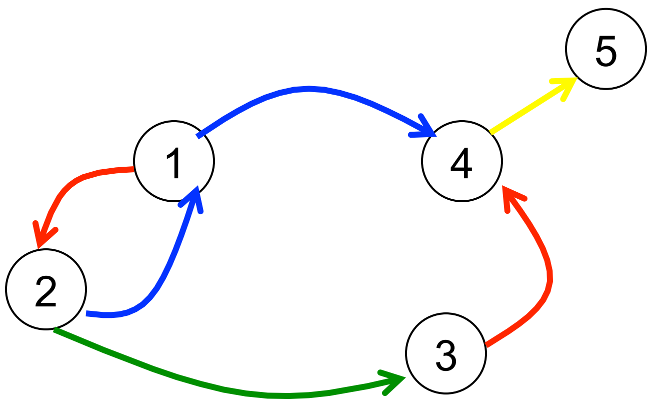
Non-linear

Discuss pros/cons in class

Simple datalog programs

Compute TC (ignoring color):

R encodes a colored graph



R=

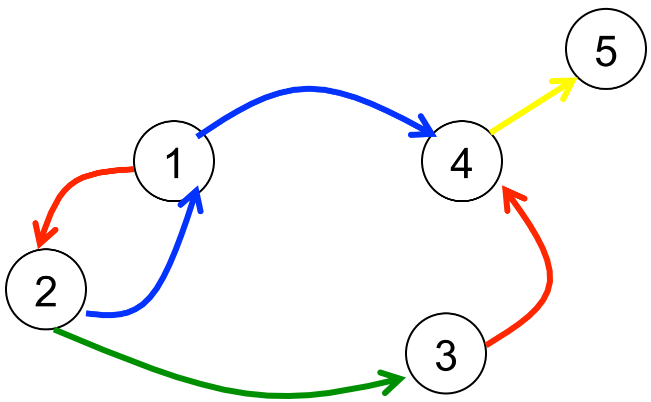
1	Red	2
2	Blue	1
2	Green	3
1	Blue	4
3	Red	4
4	Yellow	5

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

Simple datalog programs

Compute TC (ignoring color):

R encodes a colored graph



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

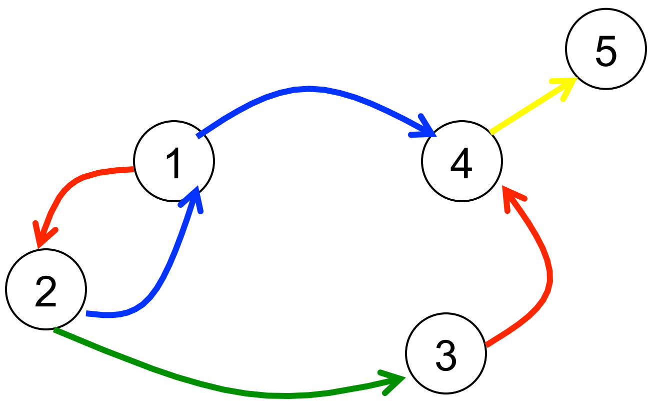
Compute pairs of nodes connected by the same color (e.g. (2,4))

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):

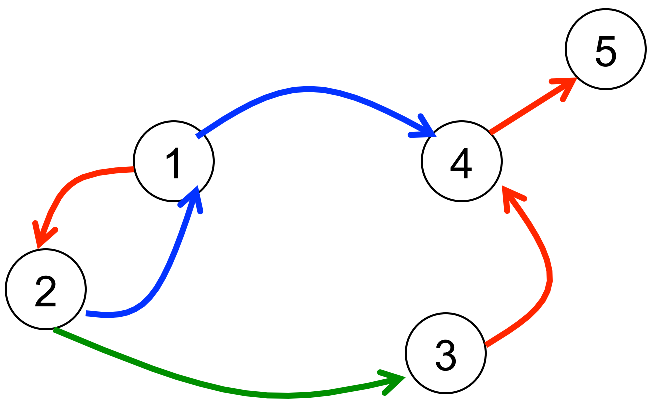
```
T(x,y) :- R(x,c,y)
T(x,y) :- R(x,c,z), T(z,y)
```

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=

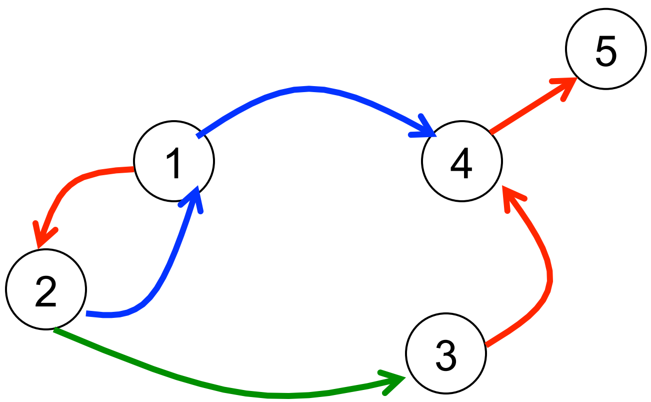
2	1
1	4

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



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)
```

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 :- $\text{body}_{11} \cup \text{body}_{12} \cup \ldots$
P_2 :- $\text{body}_{21} \cup \text{body}_{22} \cup \ldots$
….

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)

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

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:

```
P1 = P2 = … = ∅
Loop
     NewP1 = SPJU1; NewP2 = SPJU2; …
     if (NewP1 = P1 and NewP2 = P2 and …)
          then exit
     P1 = NewP1; P2 = NewP2; …
Endloop
```

Example:

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

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

```
T= ∅
Loop
     NewT(x,y) = R(x,y) ∪ Πxy(R(x,z) ⋈ 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?

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

Background: Incremental View Maintenace

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

$\boxed{V \text{ :- body}}$

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: $V \leftarrow V \cup \Delta V$

Incremental view maintenance:
Compute ΔV without having to recompute V

Background: Incremental View Maintenace

Example 1:

$\boxed{V(x,y) \text{ :- } R(x,z),S(z,y)}$

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

Background: Incremental View Maintenace

Example 1:

$\boxed{V(x,y) :- R(x,z),S(z,y)}$

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

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

Background: Incremental View Maintenace

Example 2:

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

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

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:

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

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 $\Delta V(x,y)$?

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

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
    $\Delta P_1 = \Delta\, SPJU_1$; $\Delta P_2 = \Delta SPJU_2$; …
    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-)$SPJU_i$.

```
P₁ = ΔP₁ = non-recursive-SPJU₁, P₂ = ΔP₂ = non-recursive-SPJU₂, …
Loop
    ΔP₁ = Δ SPJU₁; ΔP₂ = ΔSPJU₂;  …
    if (ΔP₁ = ∅ and ΔP₂ = ∅ and  …)
        then break
    P₁ = P₁ ∪ ΔP₁; P₂ = P₂ ∪ ΔP₂;  …
Endloop
```

Example:

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

```
T= ΔT = ? (non-recursive rule)
Loop
    ΔT(x,y)  =  ? (recursive Δ-rule)
    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
  $\Delta P_1 = \Delta\, SPJU_1$; $\Delta P_2 = \Delta SPJU_2$; …
  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)

$T(x,y) = R(x,y)$, $\Delta T(x,y) = R(x,y)$
Loop
  $\Delta T(x,y) = R(x,z), \Delta T(z,y)$
  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
    $\Delta P_1 = \Delta\, SPJU_1$; $\Delta P_2 = \Delta SPJU_2$; …
    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:

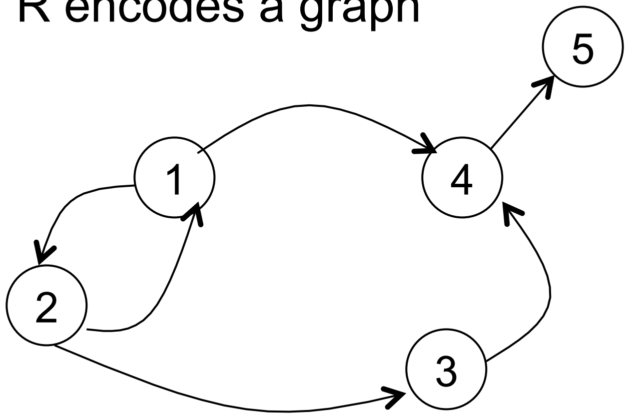
T(x,y) :- R(x,y)
T(x,y) :- R(x,z), T(z,y)

$T(x,y) = R(x,y)$, $\Delta T(x,y) = R(x,y)$
Loop
    $\Delta T(x,y) = R(x,z), \Delta T(z,y)$
    if ($\Delta T = \varnothing$)
        then break
    $T = T \cup \Delta T$
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= R,  ΔT = R
Loop
   ΔT(x,y)  = R(x,z),  ΔT(z,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

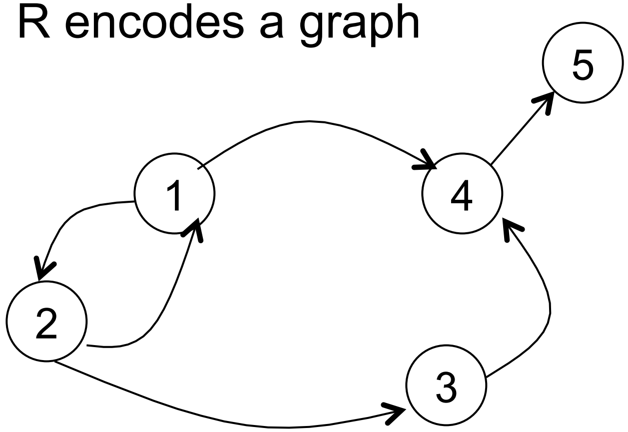
T=

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

Simple datalog programs

T= R, ΔT = R
Loop
  ΔT(x,y) = R(x,z), ΔT(z,y)
  if (ΔT = ∅)
    then break
  T = T ∪ ΔT
Endloop

R encodes a graph



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

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

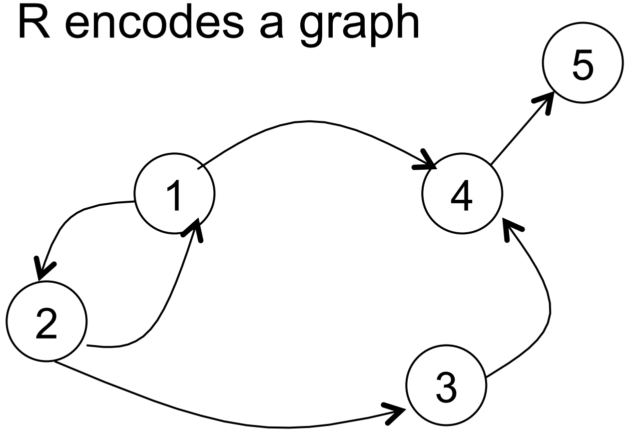
1	1
1	3
1	5
2	2
2	4
3	5

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= R,  ΔT = R
Loop
   ΔT(x,y)  = R(x,z),  ΔT(z,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

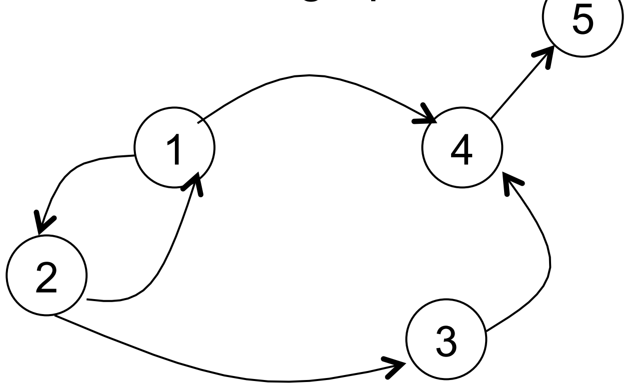
1	2
1	4
2	1
2	3
2	5

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= R,  ΔT = R
Loop
   ΔT(x,y)  = R(x,z),  ΔT(z,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 of Datalog

- Simple syntax for expressing queries with recursion
- Bottom-up evaluation – always terminates
  - Naïve evaluation
  - Semi-naïve evaluation
- Next lecture:
  - Discuss the paper on datalog
  - Datalog semantics
  - Datalog with negation