CSE544: Principles of Database Systems

Part III: Database Theory

Datalog

Why DB Theory

- Discuss the urgency of parallelism in the paper
- Discuss why data-centric approach to parallel computing
- Theory is key for understanding this approach

Outline of DB Theory

- Datalog – this lecture
- Query complexity – next week
- Static analysis (query equivalence)
- Advanced optimizations (semijoin reduction)

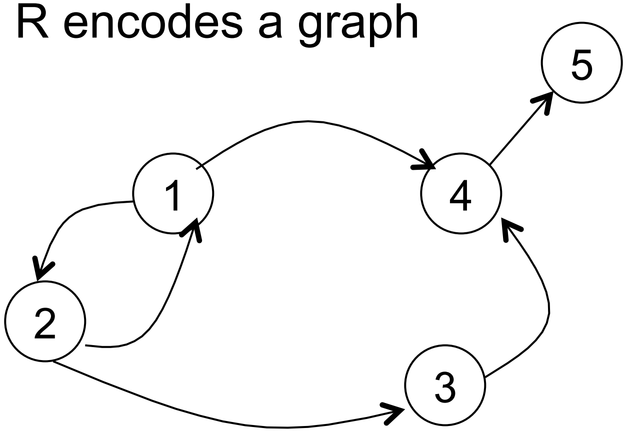
Datalog

Review the following basic concepts from Lecture 2:

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

Simple datalog programs

R encodes a graph



R=

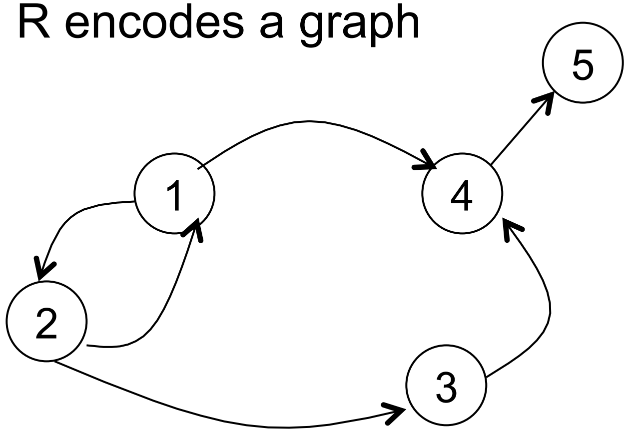
1	2
2	1
2	3
1	4
3	4
4	5

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?

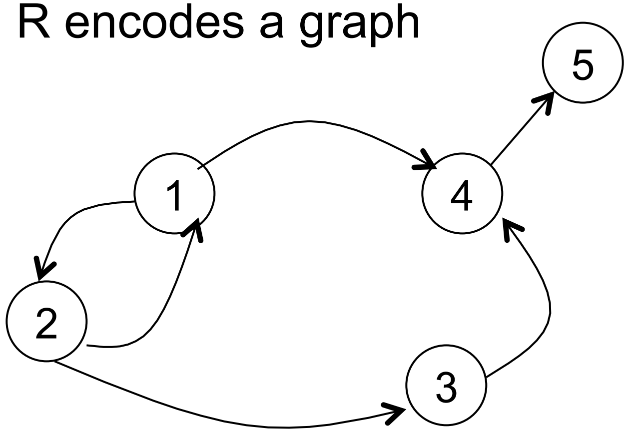
R=

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) :- 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

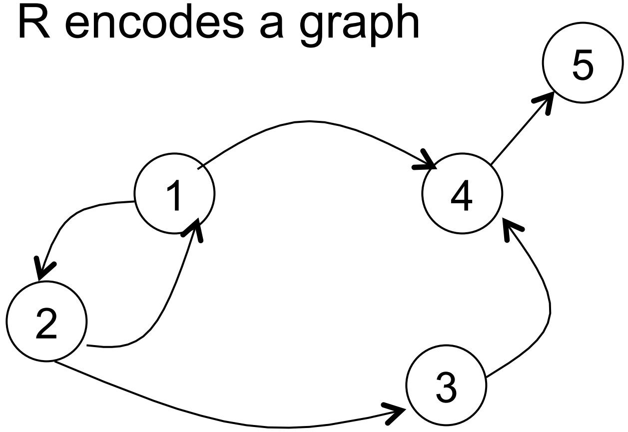
Initially:
T is empty.

First iteration:
T =

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

Simple datalog programs

R encodes a graph



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

What does it compute?

R=

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

Initially:
T is empty.

First iteration:
T =

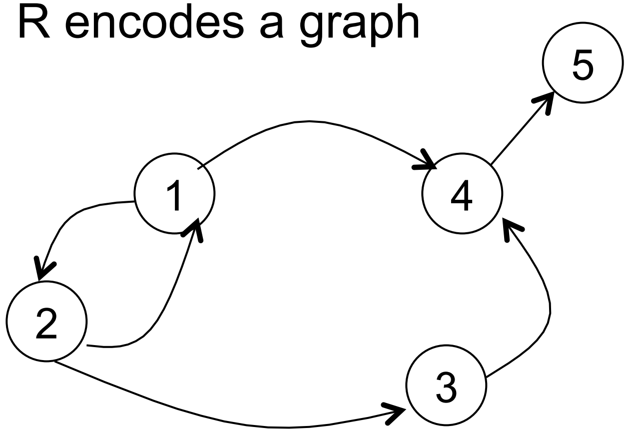
1	2
2	1
2	3
1	4
3	4
4	5

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

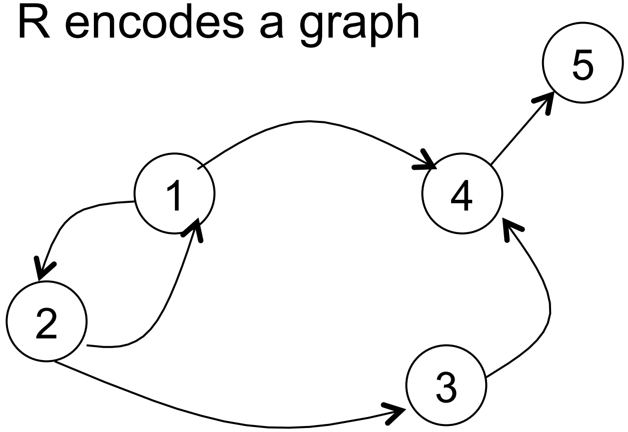
Third iteration:
T =

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

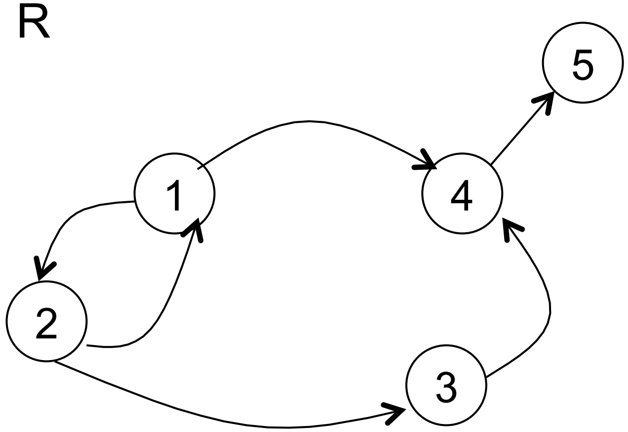
Discovered 3 times!

Discovered twice

Done

Simple datalog programs

R



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)

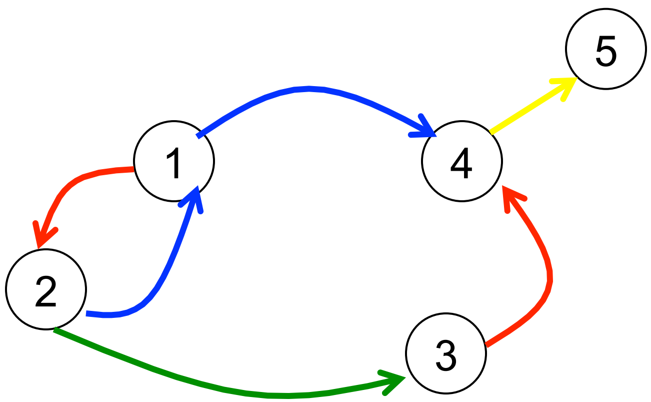
Non-linear

Discuss pros/cons in class

Simple datalog programs

Compute TC (ignoring color):

R encodes a colored graph



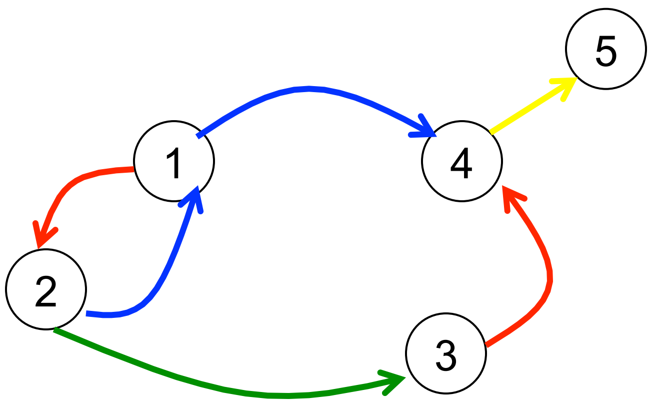
Compute pairs of nodes connected by the same color (e.g. (2,4))

R=

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

Simple datalog programs

R encodes a colored graph



R=

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

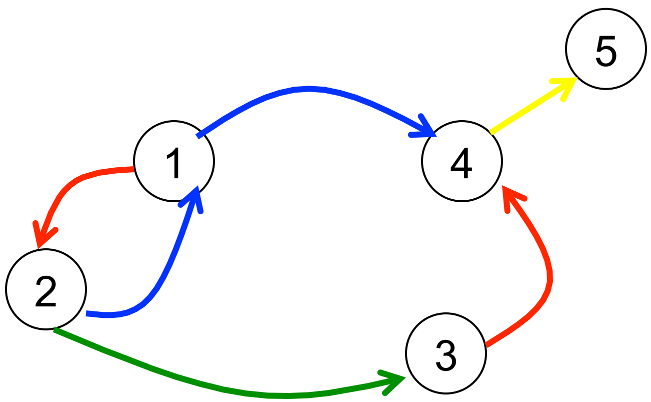
Compute TC (ignoring color):

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))

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):

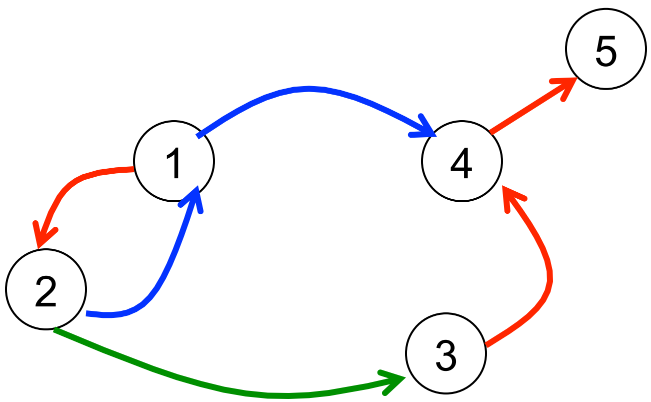
```
T(x,y) :- R(x,c,y)
T(x,y) :- R(x,c,z), T(z,y)
```

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

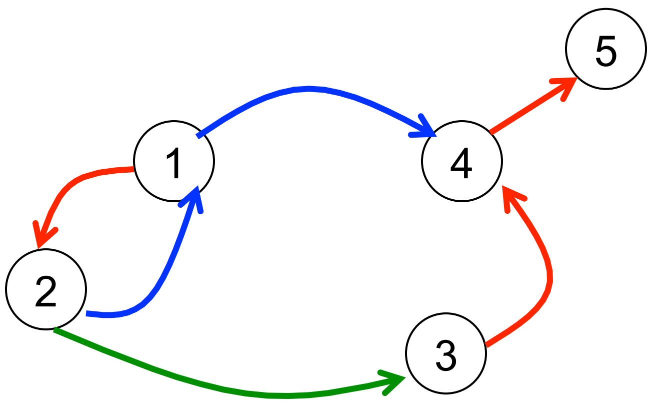
What does this program compute in general?

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

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

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

- $S = ((R \text{ or } G).B^+)^*$
- $T = ((R \text{ or } G).B^+)^*.(R \text{ or } G)$

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} :- $body_2$
….

➔ 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 break
  $P_1 = NewP_1$; $P_2 = NewP_2$; …
Endloop

Example:

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

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

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

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)

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)

Background: Incremental View Maintenace

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

Incremental view maintenance: compute ΔV (without having to recompute V)

Solution: by examples…

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

W(x,y) :- R(x,z),R(z,y) ΔW(x,y) :- ???

W(x,y) :- R(x,u),S(u,v),T(v,y) . . . .

Background: Incremental View Maintenace

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

Incremental view maintenance: compute ΔV (without having to recompute V)

Solution: by examples…

V(x,y) :- R(x,z),S(z,y) — ΔV(x,y) :- ???

W(x,y) :- R(x,z),R(z,y) — ΔW(x,y) :- ???

W(x,y) :- R(x,u),S(u,v),T(v,y) — . . . .

Background: Incremental View Maintenace

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

Incremental view maintenance: compute ΔV (without having to recompute V)

Solution: by examples…

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

ΔV(x,y) :- R(x,z), ΔS(z,y)
ΔV(x,y) :- ΔR(x,z), S(z,y)
ΔV(x,y) :- ΔR(x,z), ΔS(z,y)

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

ΔW(x,y) :- R(x,z), ΔR(z,y)
ΔW(x,y) :- ΔR(x,z), R(z,y)
ΔW(x,y) :- ΔR(x,z), ΔR(z,y)

W(x,y) :- R(x,u),S(u,v),T(v,y)

. . . .

Note: one rule may generate multiple Δ-rules

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-)$SPJU_i$.

$P_1 = P_2 = \ldots = \varnothing$,
ΔP_1 = non-recursive-$SPJU_1$, ΔP_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)
```

$T = \varnothing$, $\Delta T = R$
Loop
  $\Delta T(x,y) = \Pi_{xy}(R(x,z) \bowtie \Delta T(z,y))$
  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!

Discussion in Class

How would *you* compute the transitive closure of a very large graph R(x,y)?

- Assume a single server
- Assume a shared nothing architecture

Right linear TC

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

Non-linear TC

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

Discussion in Class

The *Declarative Imperative* paper:

- What are the extensions to datalog in Dedalus?
- What is the main usage of Dedalus described in the paper? Discuss some applications, discuss what’s missing.

Semantics of a Datalog Program

Three different, equivalent semantics:

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

Minimal Model Semantics (1/2)

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 (1/2)

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.

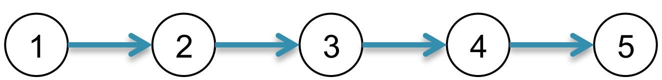
Minimal Model Semantics (1/2)

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

<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. $(I,J) \vDash \Sigma_P$

<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

T=

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

T=

1	2
1	3
1	4
1	5
2	…
…	…
…	…
5	3
5	4

All pairs of distinct nodes

Minimal Fixpoint Semantics (1/2)

Definition. Fix an EDB I, and a datalog program **P**.
The *immediate consequence* 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.

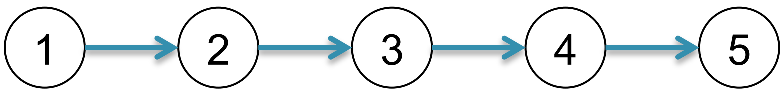
Fact. For any datalog program P, the immediate consequence operator is monotone. In other words, if $J_1 \subseteq J_2$ then $T_{\mathbf{P}}(J_1) \subseteq T_{\mathbf{P}}(J_2)$.

Theorem. The immediate consequence operator has a unique, minimal fixpoint J: $\mathrm{fix}(T_{\mathbf{P}}) = J$, 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:

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

Minimal Fixpoint Semantics (2/2)



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

R=

1	2
2	3
3	4
4	5

$J_0 = \varnothing$

T =

$J_1 = T_{\mathbf{P}}(J_0)$

1	2
2	3
3	4
4	5

$J_2 = T_{\mathbf{P}}(J_1)$

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

$J_3 = T_{\mathbf{P}}(J_2)$

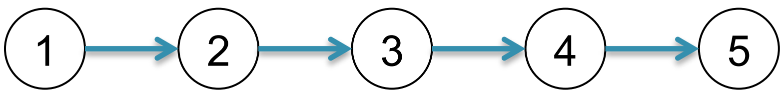
1	2
2	3
3	4
4	5
1	3
2	4
3	5
1	4
2	5

$J_4 = T_{\mathbf{P}}(J_3)$

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.

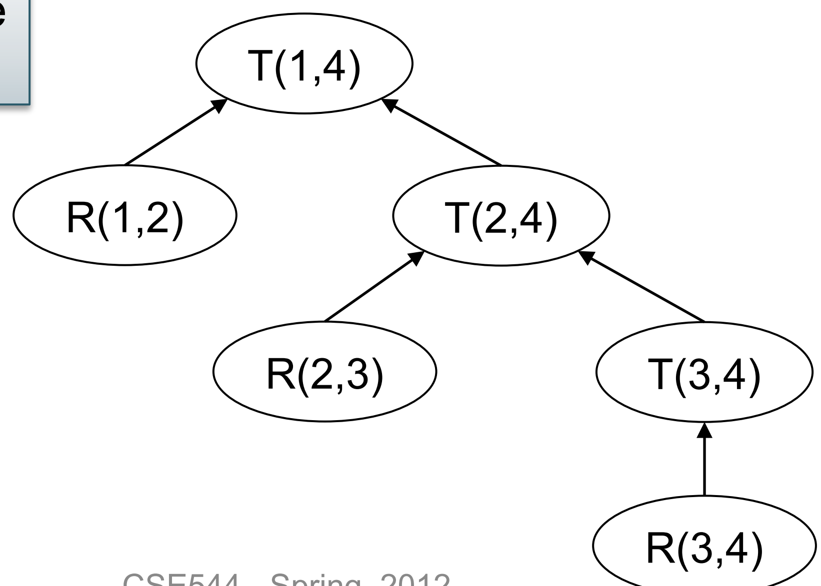


T(x,y) :- R(x,y)
T(x,y) :- R(x,z), T(z,y)

R=

1	2
2	3
3	4
4	5

Derivation tree of T(1,4)



Adding Negation: Datalog$^\neg$

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

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

Yes

Yes

There is no <u>*minimal*</u> 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 $J \cup T_{\mathbf{P}}(J)$
  – Always terminates (why ?)
- **Solution 3: Partial-fixpoint Datalog$^{\neg,*}$**
  – Compute the fixpoint of $T_{\mathbf{P}}(J)$
  – May not terminate

Stratified datalog$^\neg$

A datalog$^\neg$ 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$^\neg$ 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$^\neg$

- Evaluation algorithm for stratified datalog$^\neg$:

- 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$^{\neg}$

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

- Evaluation for Inflationary-fixpoint datalog$^{\neg}$
- 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

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¬

Discussion

Comparing datalog¬

- Compute the complement of the transitive closure in inflationary datalog¬

- Compare the expressive power of stratified datalog¬ and inflationary datalog¬

You will answer both these questions in HW3!