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

\[
\begin{align*}
R &= \{(1, 2), (2, 1), (2, 3), (1, 4), (3, 4), (4, 5)\}
\end{align*}
\]

\[
\begin{align*}
T(x, y) &\leftarrow R(x, y) \\
T(x, y) &\leftarrow R(x, z), T(z, y)
\end{align*}
\]

What does it compute?
Simple datalog programs

R encodes a graph

\[
\begin{align*}
R &= \left( \begin{array}{cc} 
1 & 2 \\
2 & 1 \\
2 & 3 \\
1 & 4 \\
3 & 4 \\
4 & 5 \\
\end{array} \right) \\
T(x,y) &\leftarrow R(x,y) \\
T(x,y) &\leftarrow R(x,z), T(z,y)
\end{align*}
\]

What does it compute?

Initially: T is empty.
Simple datalog programs

R encodes a graph

\[
R =
\begin{array}{cc}
1 & 2 \\
2 & 1 \\
2 & 3 \\
1 & 4 \\
3 & 4 \\
4 & 5 \\
\end{array}
\]

Initially: T is empty.

First iteration:
\[
T =
\begin{array}{cc}
1 & 2 \\
2 & 1 \\
2 & 3 \\
1 & 4 \\
3 & 4 \\
4 & 5 \\
\end{array}
\]

\[
T(x, y) \leftarrow R(x, y)
\]

\[
T(x, y) \leftarrow R(x, z), T(z, y)
\]

What does it compute?
Simple datalog programs

R encodes a graph

R =

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
</tr>
</tbody>
</table>

Initially: T is empty.

First iteration:

T =

```
1 2
2 1
2 3
1 4
3 4
```

Second iteration:

```
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

R =

\[
\begin{array}{|c|c|}
\hline
1 & 2 \\
2 & 1 \\
2 & 3 \\
1 & 4 \\
3 & 4 \\
4 & 5 \\
\hline
\end{array}
\]

Initially: T is empty.

First iteration:
\[
T =
\begin{array}{|c|c|}
\hline
1 & 2 \\
2 & 1 \\
2 & 3 \\
1 & 4 \\
3 & 4 \\
4 & 5 \\
\hline
\end{array}
\]

Second iteration:
\[
T =
\begin{array}{|c|c|}
\hline
1 & 2 \\
2 & 1 \\
2 & 3 \\
1 & 4 \\
3 & 4 \\
4 & 5 \\
\hline
1 & 1 \\
2 & 2 \\
1 & 3 \\
2 & 4 \\
1 & 5 \\
3 & 5 \\
\hline
\end{array}
\]

Third iteration:
\[
T =
\begin{array}{|c|c|}
\hline
1 & 2 \\
2 & 1 \\
2 & 3 \\
1 & 4 \\
3 & 4 \\
4 & 5 \\
\hline
1 & 1 \\
2 & 2 \\
1 & 3 \\
2 & 4 \\
1 & 5 \\
3 & 5 \\
\hline
2 & 5 \\
\hline
\end{array}
\]

What does it compute?

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

Initially:
T is empty.

First iteration:
T =

Second iteration:
T =

Third iteration:
T =

Discovered 3 times!

Discovered twice

Done
Simple datalog programs

Alternative ways to compute TC:

- Right linear:
  \[ T(x,y) \leftarrow R(x,y) \]
  \[ T(x,y) \leftarrow R(x,z), T(z,y) \]

- Left linear:
  \[ T(x,y) \leftarrow R(x,y) \]
  \[ T(x,y) \leftarrow T(x,z), R(z,y) \]

- Non-linear:
  \[ T(x,y) \leftarrow R(x,y) \]
  \[ T(x,y) \leftarrow T(x,z), T(z,y) \]

Discuss pros/cons in class
Simple datalog programs

 où encodes a colored graph

\[ R = \begin{array}{ccc}
1 & \text{Red} & 2 \\
2 & \text{Blue} & 1 \\
2 & \text{Green} & 3 \\
1 & \text{Blue} & 4 \\
3 & \text{Red} & 4 \\
4 & \text{Yellow} & 5 \\
\end{array} \]

Compute TC (ignoring color):

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

R encodes a colored graph

```
R =

<table>
<thead>
<tr>
<th></th>
<th>Red</th>
<th></th>
<th>Blue</th>
<th></th>
<th>Green</th>
<th></th>
<th>Blue</th>
<th></th>
<th>Red</th>
<th></th>
<th>Yellow</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td></td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td>4</td>
<td></td>
<td>3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td>1</td>
<td></td>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td>2</td>
<td></td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Blue</td>
<td></td>
<td>1</td>
<td></td>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td>2</td>
<td></td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>Blue</td>
<td></td>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td></td>
<td>3</td>
<td></td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>Red</td>
<td></td>
<td>4</td>
<td></td>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td>2</td>
<td></td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>Yellow</td>
<td></td>
<td>5</td>
<td></td>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td>3</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

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

\[
\begin{align*}
&\text{T}(x,y) :- \text{R}(x,c,y) \\
&\text{T}(x,y) :- \text{R}(x,c,z), \text{T}(z,y)
\end{align*}
\]

Compute TC (ignoring color):

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

\[
\begin{align*}
&\text{T}(x,c,y) :- \text{R}(x,c,y) \\
&\text{T}(x,c,y) :- \text{R}(x,c,z), \text{T}(z,c,y) \\
&\text{Answer}(x,y) :- \text{T}(x,c,y)
\end{align*}
\]

R encodes a colored graph

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Red</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>Blue</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>Green</td>
<td>3</td>
</tr>
<tr>
<td>1</td>
<td>Blue</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>Red</td>
<td>4</td>
</tr>
<tr>
<td>4</td>
<td>Yellow</td>
<td>5</td>
</tr>
</tbody>
</table>
Simple datalog programs

R, G, B encodes a 3-colored graph

What does this program compute in general?

\[
\begin{align*}
T(x, y) & :- R(x, y) \\
T(x, y) & :- G(x, y) \\
S(x, y) & :- T(x, z), B(z, y) \\
S(x, y) & :- S(x, z), B(z, y) \\
T(x, y) & :- S(x, z), R(z, y) \\
T(x, y) & :- S(x, z), G(z, y) \\
\text{Answer}(x, y) & :- S(x, y)
\end{align*}
\]

CSE544 - Spring, 2012
R, G, B encodes a 3-colored graph

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)

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

- S = ((R or G).B^+)*
- T = ((R or G).B^+)*.(R or G)
Syntax of Datalog Programs

The schema consists of two sets of relations:

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

A datalog program \( P \) has the form:

\[
P: \quad \begin{align*}
P_{i1}(x_{11}, & x_{12}, \ldots) : - \text{body}_1 \\
P_{i2}(x_{21}, & x_{22}, \ldots) : - \text{body}_2 \\
\ldots
\end{align*}
\]

- 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:

\[ P_{i_1} :\text{-} \text{body}_1 \]
\[ P_{i_2} :\text{-} \text{body}_2 \]

\[ \Rightarrow \]

Group by IDB predicate

\[ P_1 :\text{-} \text{body}_{11} \cup \text{body}_{12} \cup \ldots \]
\[ P_2 :\text{-} \text{body}_{21} \cup \text{body}_{22} \cup \ldots \]

\[ \Rightarrow \]

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

Naïve datalog evaluation algorithm:

\[ P_1 = P_2 = \ldots = \emptyset \]

Loop

New\( P_1 \) = SPJU\( U_1 \); New\( P_2 \) = SPJU\( U_2 \); \ldots

if (New\( P_1 \) = \( P_1 \) and New\( P_2 \) = \( P_2 \) and \ldots)

then break

\[ P_1 = \text{New} P_1; P_2 = \text{New} P_2; \ldots \]

Endloop

Example:

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

\[ \Rightarrow \]

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

\[ T= \emptyset \]

Loop

New\( T(x,y) \) = \( R(x,y) \cup \Pi_{xy}(R(x,z) \bowtie T(z,y)) \)

if (New\( T \) = \( T \))

then break

\( T = \text{New} T \)

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 Maintenance

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

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 $\Delta V$ (without having to recompute $V$)
Let $V$ be a view computed by one datalog rule (no recursion)

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 $\Delta V$ (without having to recompute $V$)

Solution: by examples…

\[
V(x,y) \leftarrow R(x,z), S(z,y)
\]

\[
\Delta V(x,y) \leftarrow ???
\]

\[
W(x,y) \leftarrow R(x,z), R(z,y)
\]

\[
\Delta W(x,y) \leftarrow ???
\]

\[
W(x,y) \leftarrow R(x,u), S(u,v), T(v,y)
\]

\ldots
Background: Incremental View Maintenance

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

$$V \leftarrow \text{body}$$

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 $\Delta V$ (without having to recompute $V$)

**Solution**: by examples…

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

- $W(x,y) \leftarrow R(x,z), R(z,y)$
  - $\Delta W(x,y) \leftarrow ???$

- $W(x,y) \leftarrow R(x,u), S(u,v), T(v,y)$
  - …
Background: Incremental View Maintenance

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

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 $\Delta V$ (without having to recompute $V$)

**Solution**: by examples...

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

\[
\Delta W(x,y) :- R(x,z), \Delta R(z,y)
\]
\[
\Delta W(x,y) :- \Delta R(z,y), R(z,y)
\]
\[
\Delta W(x,y) :- \Delta R(z,y), \Delta R(z,y)
\]

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

Note: one rule may generate multiple $\Delta$-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$_i$.

<table>
<thead>
<tr>
<th>$P_1 = P_2 = \ldots = \emptyset$,</th>
<th>$\Delta P_1 = \text{non-recursive-SPJU}_1$, $\Delta P_2 = \text{non-recursive-SPJU}_2$, ...</th>
</tr>
</thead>
<tbody>
<tr>
<td>Loop</td>
<td>$\Delta P_1 = \Delta \text{SPJU}_1$; $\Delta P_2 = \Delta \text{SPJU}_2$; ...</td>
</tr>
<tr>
<td></td>
<td>if ($\Delta P_1 = \emptyset$ and $\Delta P_2 = \emptyset$ and ...) then break</td>
</tr>
<tr>
<td></td>
<td>$P_1 = P_1 \cup \Delta P_1$; $P_2 = P_2 \cup \Delta P_2$; ...</td>
</tr>
</tbody>
</table>

Example:

| $T(x,y) \leftarrow R(x,y)$ | $T(x,y) \leftarrow R(x,z)$, $T(z,y)$ |

Note: for any linear datalog programs, the semi-naïve algorithm has only one $\Delta$-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

\[
\begin{align*}
T(x,y) & : \neg R(x,y) \\
T(x,y) & : \neg R(x,z), T(z,y)
\end{align*}
\]

Non-linear TC

\[
\begin{align*}
T(x,y) & : \neg R(x,y) \\
T(x,y) & : T(x,z), T(z,y)
\end{align*}
\]
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 $r$: $P(x_1\ldots x_k) :- R_1(...), R_2(...), ...$

Associate the logical sentence $\Sigma_r$: $\forall z_1\ldots \forall z_n. ((R_1(...) \land R_2(...) \land ...) \Rightarrow P(...)]$

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

**Definition.** If $P$ is a datalog program, $\Sigma_P$ is the set of all logical sentences associated to its rules.

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

Sentence: $\forall x. \forall y. \forall z.(R(x,z) \land T(z,y) \Rightarrow T(x,y)$

$\equiv \forall x. \forall y.(\exists z. R(x,z) \land T(z,y) \Rightarrow T(x,y)$
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)**

**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:**

\[
\begin{array}{c}
1 \\
2 \\
3 \\
4 \\
5 \\
\end{array}
\]

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

\[
\begin{array}{|c|c|}
\hline
R & \hline
1 & 2 \\
2 & 3 \\
3 & 4 \\
4 & 5 \\
\hline
\end{array}
\]

\[
\begin{array}{|c|c|}
\hline
T & \hline
1 & 2 \\
2 & 3 \\
3 & 4 \\
4 & 5 \\
\hline
\end{array}
\]

\[
\begin{array}{|c|c|}
\hline
T(x,y) & \hline
T(x,y) & :- R(x,y) \\
T(x,y) & :- R(x,z), T(z,y) \\
\hline
\end{array}
\]

---

**All pairs of distinct nodes**
**Definition.** Fix an EDB I, and a datalog program P. The *immediate consequence* operator $T_P$ is defined as follows.
For any IDB J:

$$T_P(J) = \text{all IDB facts that are immediate consequences from I and J.}$$

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

**Theorem.** The immediate consequence operator has a unique, minimal fixpoint J:

$$\text{fix}(T_P) = J,$$
where J is the minimal instance with the property $T_P(J) = J$.

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

$$\text{fix } (T_P) = J_0 \cup J_1 \cup J_2 \cup \ldots \quad \text{where} \quad J_0 = \emptyset, \quad J_{k+1} = T_P(J_k)$$
Minimal Fixpoint Semantics (2/2)

\[
R = \begin{array}{c|c}
1 & 2 \\
2 & 3 \\
3 & 4 \\
4 & 5 \\
\end{array}
\]

\[
T = \begin{array}{c|c}
\end{array}
\]

\[
J_0 = \emptyset
\]

\[
J_1 = T P(J_0)
\]

\[
J_2 = T P(J_1)
\]

\[
J_3 = T P(J_2)
\]

\[
J_4 = T P(J_3)
\]

\[
T(x,y) :- R(x,y)
\]

\[
T(x,y) :- R(x,z), T(z,y)
\]
Proof Theoretic Semantics

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

\[ R = \begin{array}{cc}
1 & 2 \\
2 & 3 \\
3 & 4 \\
4 & 5 \\
\end{array} \]

\[
T(x,y) : - R(x,y) \\
T(x,y) : - R(x,z), T(z,y)
\]
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 \( P \)?

\[ J_1 = \{ \} \quad J_2 = \{ S(a) \} \quad J_3 = \{ T(a) \} \quad J_4 = \{ S(a), T(a) \} \]
Recursion and Negation

Don’t Like Each Other

EDB: \( I = \{ R(a) \} \)

\[ S(x) : R(x), \text{not } T(x) \]
\[ T(x) : R(x), \text{not } S(x) \]

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 \( R(a), S(a), \lnot T(a) \) and both rules are true.
Yes
Yes

There is no \textit{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?

J₁ = { }  Yes: the facts in J₂ are R(a), S(a), ¬T(a) and both rules are true.
J₂ = {S(a)}  No: both rules fail
J₃ = {T(a)}  Yes
J₄ = {S(a), T(a)}  Yes

There is no minimal model!
(Why does Knaster-Tarski’s theorem fail?)
Adding Negation: $\text{datalog}^\neg$

- **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 $\leq k$ may be negated in strata $\geq k+1$

- **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
A datalog\(^{-}\) program is \textit{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, \(\ldots\), \(i-1\).

\[ P_1 :\text{- body}_1 \]
\[ P_2 :\text{- body}_2 \]
\[ \quad \ldots \]
\[ \quad \ldots \]
\[ \quad \ldots \]
\[ \quad \ldots \]
\[ P_j :\text{- body}_j \]
\[ \quad \quad \ldots \]
\[ \quad \quad \ldots \]
\[ \quad \quad \ldots \]
\[ \quad \quad \ldots \]
\[ P_n :\text{- body}_n \]

\[ \text{Stratum } i \]

Note: a datalog\(^{-}\) program either is stratified or it ain’t!

Which programs are stratified?

\[ T(x,y) :\text{- } R(x,y) \]
\[ T(x,y) :\text{- } T(x,z), R(z,y) \]
\[ CT(x,y) :\text{- } \text{Node}(x), \text{Node}(y), \text{not } T(x,y) \]

\[ S(x) :\text{- } R(x), \text{not } T(x) \]
\[ T(x) :\text{- } R(x), \text{not } S(x) \]
Stratified datalog⁻

- Evaluation algorithm for stratified datalog⁻:
  - For each stratum \( i = 1, 2, \ldots \), 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?

---

\[
\begin{align*}
  T(x,y) & : \text{-} R(x,y) \\
  T(x,y) & : T(x,z), R(z,y) \\
  \text{CT}(x,y) & : \text{Node}(x), \text{Node}(y), \text{not} \ T(x,y)
\end{align*}
\]
Stratified datalog$^-$

• Evaluation algorithm for stratified datalog$^-$:
  
  • For each stratum $i = 1, 2, \ldots, \text{do}$:
    
    – 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:

$J_1 = \{ T = \text{transitive closure, CT = its complement} \}$

$J_2 = \{ T = \text{all pairs of nodes, CT = empty} \}$

\[
\begin{align*}
T(x,y) & ::= R(x,y) \\
T(x,y) & ::= T(x,z), R(z,y) \\
CT(x,y) & ::= \text{Node}(x), \text{Node}(y), \text{not } T(x,y)
\end{align*}
\]
Inflationary-fixpoint datalog

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

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

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

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

Find the inflationary semantics for:

- $T(x,y) :- R(x,y)$
- $T(x,y) :- T(x,z), R(z,y)$
- $CT(x,y) :- \text{Node}(x), \text{Node}(y), \text{not} \ T(x,y)$

- $S(x) :- R(x), \text{not} \ T(x)$
- $T(x) :- R(x), \text{not} \ S(x)$
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^-,*

Let \( P \) be any datalog^- program, and \( I \) an EDB.
Let \( T_P(J) \) be the *immediate consequence* operator.

Define the sequence: \( J_0 = \emptyset, J_{n+1} = T_P(J_n), \) 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:

\[
\begin{align*}
T(x,y) & :- R(x,y) \\
T(x,y) & :- T(x,z), R(z,y) \\
CT(x,y) & :- Node(x), Node(y), \text{not } T(x,y)
\end{align*}
\]

\[
\begin{align*}
S(x) & :- R(x), \text{not } T(x) \\
T(x) & :- R(x), \text{not } S(x)
\end{align*}
\]

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