Announcement

• HW1 due this Friday

• Project proposals M2 due next Friday
  – Project feedback meetings February 7
  – Will meet on different day with people who go to the ski day

• Paper review due next Wednesday
Where We Are

Relational query languages:

• SQL
• Relational Algebra
• Relational Calculus (haven’t discussed, but you may look it up)

They can express the same class of queries called \textit{relational queries}
Which are Relational Queries? Which are not? And Why?

Friend(X,Y)

• Find all people X whose number of friends is a prime number
• Find all people who are friends with everyone who is not a friend of Bob
• Partition all people into three sets $P_1(X), P_2(X), P_3(X)$ s.t. any two friends are in different partitions
• Find all people who are direct or indirect friends with Alice
Which are Relational Queries? Which are not? And Why?

Friend(X,Y)

• Find all people X whose number of friends is a prime number

No higher math in database
Which are Relational Queries? Which are not? And Why?

Friend(X,Y)

• Find all people X whose number of friends is a prime number

• Find all people who are friends with everyone who is not a friend of Bob
Which are Relational Queries? Which are not? And Why?

Friend(X,Y)

- Find all people X whose number of friends is a prime number
- Find all people who are friends with everyone who is not a friend of Bob

Yes! (write it in SQL!)
Which are Relational Queries? Which are not? And Why?

Friend(X,Y)

• Find all people X whose number of friends is a prime number
• Find all people who are friends with everyone who is not a friend of Bob

• Partition all people into three sets P1(X), P2(X), P3(X) s.t. any two friends are in different partitions
Which are Relational Queries? Which are not? And Why?

Friend(X,Y)

• Find all people X whose number of friends is a prime number
• Find all people who are friends with everyone who is not a friend of Bob

• Partition all people into three sets P1(X), P2(X), P3(X) s.t. any two friends are in different partitions

No! NP-complete
Which are Relational Queries? Which are not? And Why?

Friend(X,Y)

• Find all people X whose number of friends is a prime number
• Find all people who are friends with everyone who is not a friend of Bob

• Partition all people into three sets P1(X), P2(X), P3(X) s.t. any two friends are in different partitions
• Find all people who are direct or indirect friends with Alice
Which are Relational Queries? Which are not? And Why?

Friend(X,Y)
• Find all people X whose number of friends is a prime number
• Find all people who are friends with everyone who is not a friend of Bob

“Recursive query”; PTIME, yet not expressible in RA

• Partition all people into three sets P1(X), P2(X), P3(X) s.t. any two friends are in different partitions
• Find all people who are direct or indirect friends with Alice
Recursive Queries

• “Find all direct or indirect friends of Alice”
• Computable in PTIME, yet not expressible in RA
• Datalog: extends RA with recursive queries
Datalog

- Designed in the 80’s
- Simple, concise, elegant
- Today is a hot topic, beyond databases: network protocols, static program analysis, DB+ML
- Very few open source implementations, and hard to find
- In HW2 we will use Souffle
Outline

• Datalog rules
• Recursion
• Semantics
• Negation, aggregates, stratification
• Naïve and Semi-naïve Evaluation
Datalog: Facts and Rules

Schema

Actor(id, fname, lname)
Casts(pid, mid)
Movie(id, name, year)
Datalog: Facts and Rules

Facts = tuples in the database

Rules = queries
Datalog: Facts and Rules

Facts = tuples in the database

Rules = queries

Actor(id, fname, lname)
Casts(pid, mid)
Movie(id, name, year)

Actor(344759, ‘Douglas’, ‘Fowley’).
Casts(344759, 29851).
Casts(355713, 29000).
Movie(29445, ‘Ave Maria’, 1940).
Datalog: Facts and Rules

**Facts** = tuples in the database

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

**Rules** = queries

Q1(y) :- Movie(x,y,z), z='1940'.
Datalog: Facts and Rules

**Facts** = tuples in the database

- Actor(344759, ‘Douglas’, ‘Fowley’).
- Casts(344759, 29851).
- Casts(355713, 29000).

**Rules** = queries

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

Find Movies made in 1940
Datalog: Facts and Rules

**Facts** = tuples in the database

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

**Rules** = queries

- Q1(y) :- Movie(x,y,z), z='1940'.
- Q2(f, l) :- Actor(z,f,l), Casts(z,x), Movie(x,y,'1940').
Datalog: Facts and Rules

Facts = tuples in the database

Actor(344759, ‘Douglas’, ‘Fowley’).
Casts(344759, 29851).
Casts(355713, 29000).
Movie(29445, ‘Ave Maria’, 1940).

Rules = queries

Q1(y) :- Movie(x, y, z), z = ‘1940’.
Q2(f, l) :- Actor(z, f, l), Casts(z, x), Movie(x, y, ’1940’).

Find Actors who acted in Movies made in 1940
Actor(id, fname, lname)
Casts(pid, mid)
Movie(id, name, year)

Datalog: Facts and Rules

**Facts** = tuples in the database

Actor(344759, ‘Douglas’, ‘Fowley’).
Casts(344759, 29851).
Casts(355713, 29000).
Movie(29445, ‘Ave Maria’, 1940).

**Rules** = queries

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

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

Q3(f, l) :- Actor(z, f, l), Casts(z, x1), Movie(x1, y1, 1910),
            Casts(z, x2), Movie(x2, y2, 1940).
Datalog: Facts and Rules

**Facts** = tuples in the database

- Actor(344759, ‘Douglas’, ‘Fowley’).
- Casts(344759, 29851).
- Casts(355713, 29000).

**Rules** = queries

- Q1(y) :- Movie(x,y,z), z='1940'.
- Q2(f, l) :- Actor(z,f,l), Casts(z,x), Movie(x,y,’1940’).
- Q3(f,l) :- Actor(z,f,l), Casts(z,x1), Movie(x1,y1,1910), Casts(z,x2), Movie(x2,y2,1940)

Find Actors who acted in a Movie in 1940 and in one in 1910
**Datalog: Facts and Rules**

**Facts** = tuples in the database

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

**Rules** = queries

- Q1(y) :- Movie(x, y, z), z='1940'
- Q2(f, l) :- Actor(z, f, l), Casts(z, x), Movie(x, y, '1940')
- Q3(f, l) :- Actor(z, f, l), Casts(z, x1), Movie(x1, y1, 1910), Casts(z, x2), Movie(x2, y2, 1940)

**Extensional Database Predicates** = EDB = Actor, Casts, Movie

**Intensional Database Predicates** = IDB = Q1, Q2, Q3
Anatomy of a Rule

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

f, l = head variables
x,y,z = existential variables
More Datalog Terminology

Q(args) :- R1(args), R2(args), ...

- \( R_i(\text{args}_i) \) called an \textit{atom}, or a \textit{relational predicate}
More Datalog Terminology

Q(args) :- R1(args), R2(args), ....

- $R_i(args_i)$ called an atom, or a relational predicate
- $R_i(args_i)$ evaluates to true when relation $R_i$ contains the tuple described by $args_i$.
  - Example: Actor(344759, ‘Douglas’, ‘Fowley’) is true
More Datalog Terminology

- \( R_i(\text{args}_i) \) called an \textit{atom}, or a \textit{relational predicate}
- \( R_i(\text{args}_i) \) evaluates to true when relation \( R_i \) contains the tuple described by \( \text{args}_i \).
  - Example: \( \text{Actor}(344759, ‘Douglas’, ‘Fowley’) \) is true
- In addition we can also have arithmetic predicates
  - Example: \( z > ‘1940’ \).
More Datalog Terminology

Q(args) :- R1(args), R2(args), ....

• $R_i(args_i)$ called an **atom**, or a **relational predicate**
• $R_i(args_i)$ evaluates to true when relation $R_i$ contains the tuple described by $args_i$.
  – Example: Actor(344759, ‘Douglas’, ‘Fowley’) is true
• In addition we can also have arithmetic predicates
  – Example: $z > ‘1940’$.
• Some systems use $\leftarrow$
More Datalog Terminology

- $R_i(\text{args}_i)$ called an **atom**, or a **relational predicate**
- $R_i(\text{args}_i)$ evaluates to true when relation $R_i$ contains the tuple described by $\text{args}_i$.
  - Example: $\text{Actor}(344759, 'Douglas', 'Fowley$') is true
- In addition we can also have arithmetic predicates
  - Example: $z > '1940$'.
- Some systems use $\leftarrow$
- Some use $\text{AND}$
Semantics of a Single Rule

• Meaning of a datalog rule = a logical statement

\[ Q1(y) : - \text{Movie}(x,y,z), \ z=’1940’ . \]
Semantics of a Single Rule

- Meaning of a datalog rule = a logical statement!

\[
Q1(y) :- \text{Movie}(x,y,z), z='1940'.
\]

- If \((x,y,z) \in \text{Movies} \) and \(z = '1940'\) then \(y\) is in answer
Semantics of a Single Rule

- Meaning of a datalog rule = a logical statement!
  \[
  Q1(y) :\text{- Movie}(x,y,z), \ z='1940'.
  \]
- If \((x,y,z) \in \text{Movies}\) and \(z = '1940'\) then \(y\) is in answer
  \[
  \forall x \forall y \forall z \ [(\text{Movie}(x,y,z) \text{ and } z='1940') \Rightarrow Q1(y)]
  \]
Semantics of a Single Rule

- Meaning of a datalog rule = a logical statement!
  
  \[ Q1(y) \leftarrow \text{Movie}(x,y,z), \ z='1940'. \]

- If \((x,y,z) \in \text{Movies}\) and \(z = '1940'\) then \(y\) is in answer

  \[ \forall x \forall y \forall z \ [(\text{Movie}(x,y,z) \land z='1940') \Rightarrow Q1(y)] \]

- We want \text{smallest} answer with this property (why?)
Semantics of a Single Rule

- Meaning of a datalog rule = a logical statement!
  
  \[ Q1(y) : - \text{Movie}(x,y,z), \ z='1940'. \]

- If \((x,y,z) \in \text{Movies}\) and \(z = '1940'\) then \(y\) is in answer
  
  \[ \forall x \forall y \forall z \ [(\text{Movie}(x,y,z) \text{ and } z='1940') \Rightarrow Q1(y)] \]

- We want \textit{smallest} answer with this property (why?)

- Logically equivalent:
  
  \[ \forall y \ [(\exists x \exists z \text{Movie}(x,y,z) \text{ and } z='1940') \Rightarrow Q1(y)] \]

Actor(id, fname, lname)
Casts(pid, mid)
Movie(id, name, year)
Semantics of a Single Rule

• Meaning of a datalog rule = a logical statement!

\[
Q(x) : - \text{Movie}(x,y,z), z='1940'.
\]

• If (x,y,z) ∈ Movies and z = ‘1940’ then y is in answer

\[
\forall x \forall y \forall z \ [\text{Movie}(x,y,z) \text{ and } z='1940') \Rightarrow Q1(y)\]

• We want \textit{smallest} answer with this property (why?)

• Logically equivalent:

\[
\forall y \ [(\exists x \exists z \text{Movie}(x,y,z) \text{ and } z='1940') \Rightarrow Q1(y)]
\]

• Non-head variables are called "existential variables"
Outline

- Datalog rules
- Recursion
- Semantics
- Negation, aggregates, stratification
- Naïve and Semi-naïve Evaluation
Datalog program

• A datalog program consists of several rules
• Importantly, rules may be recursive!
• Usually there is one distinguished predicate that’s the final answer
• We will show an example first, then give the general semantics.
Example

R encodes a graph

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

R encodes a graph

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

What does it compute?
R encodes a graph

Initially:
T is empty.

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
</tr>
<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>

What does it compute?

Example

\[
R(x, y) : - R(x, y), T(x, y) \\
T(x, y) : - R(x, z), T(z, y)
\]
R encodes a graph

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
</tr>
<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 =

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
</tr>
<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>

First rule generates this

Second rule generates nothing (because T is empty)

Example

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>T(x,y) :- R(x,y)</td>
<td></td>
</tr>
<tr>
<td>T(x,y) :- R(x,z), T(z,y)</td>
<td></td>
</tr>
</tbody>
</table>
R encodes a graph

First iteration:
\[
T = \begin{pmatrix}
1 & 2 \\
2 & 1 \\
2 & 3 \\
1 & 4 \\
3 & 4 \\
4 & 5 \\
\end{pmatrix}
\]
Initially: 
\[T\text{ is empty.}\]

Second rule generates this

Second iteration:
\[
T = \begin{pmatrix}
1 & 2 \\
2 & 1 \\
2 & 3 \\
1 & 4 \\
3 & 4 \\
4 & 5 \end{pmatrix}
\]

What does it compute?

Example

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

R encodes a graph

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

What does it compute?

Initially:
\[
T \text{ is empty.}
\]

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

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

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

Both rules

First rule

Second rule

New fact
**Example**

R encodes a graph

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

Initially:
T is empty.

<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>

First iteration:
T =

<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>

Second iteration:
T =

<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>

Third iteration:
T =

<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>

Fourth iteration:
T = (same)

No new facts.

DONE
Three Equivalent Programs

R encodes a graph

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

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

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

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

Right linear
Left linear
Non-linear

Question: which terminates in fewest iterations?
Outline

- Datalog rules
- Recursion
- Semantics
  - Negation, aggregates, stratification
  - Naïve and Semi-naïve Evaluation
1. Fixpoint Semantics

- Start: $\text{IDB}_0 = \text{empty relations}$; $t = 0$
- Repeat:
  $\text{IDB}_{t+1} = \text{Compute Rules(EDB, IDB}_t)$
  $t = t+1$
- Until $\text{IDB}_t = \text{IDB}_{t-1}$
1. Fixpoint Semantics

• Start: $\text{IDB}_0 = \text{empty relations}; \ t = 0$
  Repeat:
  $\text{IDB}_{t+1} = \text{Compute Rules(EDB, IDB}_t)$
  $t = t+1$
  Until $\text{IDB}_t = \text{IDB}_{t-1}$

• Remark: since rules are monotone:
  $\emptyset = \text{IDB}_0 \subseteq \text{IDB}_1 \subseteq \text{IDB}_2 \subseteq \ldots$
1. Fixpoint Semantics

- Start: $\text{IDB}_0 = \text{empty relations}; ~ t = 0$
  
  Repeat:
  
  $\text{IDB}_{t+1} = \text{Compute Rules}(\text{EDB}, \text{IDB}_t)$
  
  $t = t+1$

  Until $\text{IDB}_t = \text{IDB}_{t-1}$

- Remark: since rules are monotone:
  
  $\emptyset = \text{IDB}_0 \subseteq \text{IDB}_1 \subseteq \text{IDB}_2 \subseteq ...$

- A datalog program w/o functions (+, *, ...) always terminates. (In what time?)
2. Minimal Model Semantics:

• Find some IDB instance that satisfies:
  1) For every rule,
     \[ \forall \text{vars} \ (\text{Body}(\text{EDB}, \text{IDB}) \Rightarrow \text{Head}(\text{IDB})] \]
  2) Is the smallest IDB satisfying (1)
2. Minimal Model Semantics:

• Find some IDB instance that satisfies:
  1) For every rule,
     \[ \forall \text{vars} \ [(\text{Body}(\text{EDB},\text{IDB}) \Rightarrow \text{Head}(\text{IDB})]\]
  2) Is the smallest IDB satisfying (1)
2. Minimal Model Semantics:

- Find some IDB instance that satisfies:
  1) For every rule,
     \[ \forall \text{vars} \, [(\text{Body}(\text{EDB},\text{IDB}) \Rightarrow \text{Head}(\text{IDB})] \]
  2) Is the smallest IDB satisfying (1)

- **Theorem:** there exists a unique such instance
2. Minimal Model Semantics:

• Find some IDB instance that satisfies:
  1) For every rule,
     \[\forall \text{vars}\ [(\text{Body}(\text{EDB},\text{IDB}) \Rightarrow \text{Head}(\text{IDB})]\]
  2) Is the smallest IDB satisfying (1)

• **Theorem**: there exists a unique such instance

• It doesn’t tell us how to find it…
2. Minimal Model Semantics:

- Find some IDB instance that satisfies:
  1) For every rule,\[\forall \text{vars} \left[ (\text{Body}(\text{EDB}, \text{IDB}) \Rightarrow \text{Head}(\text{IDB}) \right] \]
  2) Is the smallest IDB satisfying (1)

- **Theorem**: there exists a unique such instance

- It doesn’t tell us how to find it…

- …but we know how: compute fixpoint!
Example

\[
\begin{align*}
T(x,y) & : R(x,y) \\
T(x,y) & : R(x,z), T(z,y)
\end{align*}
\]
1. Fixpoint semantics:

- Start: \( T_0 = \emptyset; t = 0 \)
  
  Repeat:
  
  \[
  T_{t+1}(x,y) = R(x,y) \cup \Pi_{xy}(R(x,z) \bowtie T_t(z,y))
  \]

  \[
  t = t + 1
  \]

  Until \( T_t = T_{t-1} \)
Example

1. Fixpoint semantics:
   - Start: $T_0 = \emptyset$; $t = 0$
   - Repeat:
     $$T_{t+1}(x,y) = R(x,y) \cup \Pi_{xy}(R(x,z) \bowtie T_t(z,y))$$
     $$t = t+1$$
   - Until $T_t = T_{t-1}$

2. Minimal model semantics: smallest $T$ s.t.
   - $\forall x \forall y [(R(x,y) \Rightarrow T(x,y))] \land$
   - $\forall x \forall y \forall z [(R(x,z) \land T(z,y)) \Rightarrow T(x,y)]$
Datalog Semantics

• The fixpoint semantics tells us how to compute a datalog query

• The minimal model semantics is more declarative: only says what we get

• The two semantics are equivalent meaning: you get the same thing
Outline

• Datalog rules
• Recursion
• Semantics
  • Negation, aggregates, stratification
  • Naïve and Semi-naïve Evaluation
More Features

- Aggregates
- Grouping
- Negation
Aggregates

\[ \text{aggregate name} \ <\var> : \{ \ [\text{relation to compute aggregate on}] \} \]

\[ \text{min} \ x : \{ \ \text{Actor}(x, y, _), y = \text{‘John’} \} \]

\[ Q(\text{minId}) : - \text{minId} = \text{min} \ x : \{ \ \text{Actor}(x, y, _), y = \text{‘John’} \} \]

Assign variable to the value of the aggregate

Meaning (in SQL)

```sql
SELECT min(id) as minId
FROM Actor as a
WHERE a.name = ‘John’
```

Aggregates in Souffle:
- count
- min
- max
- sum
Counting

Q(c) :- c = count : { Actor(_, y, _), y = ‘John’ }

No variable here!

Meaning (in SQL, assuming no NULLs)

```
SELECT count(*) as c
FROM Actor as a
WHERE a.name = ‘John’
```
Grouping

\[ Q(y,c) : - \text{Movie}(_,_,y), \ c = \text{count} : \{ \text{Movie}(_,_,y) \} \]

Meaning (in SQL)

```
SELECT m.year, count(*)
FROM Movie as m
GROUP BY m.year
```
Examples

A genealogy database (parent/child)

ParentChild

<table>
<thead>
<tr>
<th>p</th>
<th>c</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alice</td>
<td>Carol</td>
</tr>
<tr>
<td>Bob</td>
<td>Carol</td>
</tr>
<tr>
<td>Bob</td>
<td>David</td>
</tr>
<tr>
<td>Carol</td>
<td>Eve</td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
</tbody>
</table>
Count Descendants

For each person, count his/her descendants
Count Descendants

For each person, count his/her descendants

Answer

<table>
<thead>
<tr>
<th>p</th>
<th>cnt</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alice</td>
<td>4</td>
</tr>
<tr>
<td>Bob</td>
<td>5</td>
</tr>
<tr>
<td>Carol</td>
<td>3</td>
</tr>
<tr>
<td>David</td>
<td>2</td>
</tr>
<tr>
<td>Fred</td>
<td>1</td>
</tr>
</tbody>
</table>
Count Descendants

For each person, count his/her descendants

Alice
Bob
Carol
David
Fred
George

Answer

<table>
<thead>
<tr>
<th>p</th>
<th>cnt</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alice</td>
<td>4</td>
</tr>
<tr>
<td>Bob</td>
<td>5</td>
</tr>
<tr>
<td>Carol</td>
<td>3</td>
</tr>
<tr>
<td>David</td>
<td>2</td>
</tr>
<tr>
<td>Fred</td>
<td>1</td>
</tr>
</tbody>
</table>

Note: Eve and George do not appear in the answer (why?)
Count Descendants

For each person, compute the total number of descendants

// for each person, compute his/her descendants
Count Descendants

For each person, compute the total number of descendants

// for each person, compute his/her descendants
D(x,y) :- ParentChild(x,y).
Count Descendants

For each person, compute the total number of descendants

// for each person, compute his/her descendants
D(x,y) :- ParentChild(x,y).
D(x,z) :- D(x,y), ParentChild(y,z).
Count Descendants

For each person, compute the total number of descendants

// for each person, compute his/her descendants
D(x,y) :- ParentChild(x,y).
D(x,z) :- D(x,y), ParentChild(y,z).

// For each person, count the number of descendants
Count Descendants

For each person, compute the total number of descendants

// for each person, compute his/her descendants
D(x,y) :- ParentChild(x,y).
D(x,z) :- D(x,y), ParentChild(y,z).

// For each person, count the number of descendants
T(p,c) :- D(p,_), c = count : { D(p,y) }. 

ParentChild(p,c)
How many descendants does Alice have?

// for each person, compute his/her descendants
D(x, y) :- ParentChild(x, y).
D(x, z) :- D(x, y), ParentChild(y, z).

// For each person, count the number of descendants
T(p, c) :- D(p, _), c = count : { D(p, y) }.
Count Descendants

How many descendants does Alice have?

// for each person, compute his/her descendants
D(x,y) :- ParentChild(x,y).
D(x,z) :- D(x,y), ParentChild(y,z).

// For each person, count the number of descendants
T(p,c) :- D(p,_), c = count : { D(p,y) }.

// Find the number of descendants of Alice
How many descendants does Alice have?

// for each person, compute his/her descendants
D(x,y) :- ParentChild(x,y).
D(x,z) :- D(x,y), ParentChild(y,z).

// For each person, count the number of descendants
T(p,c) :- D(p,_), c = count : { D(p,y) }.

// Find the number of descendants of Alice
Q(d) :- T(p,d), p = “Alice”.

ParentChild(p,c)
Find all descendants of Bob that are not descendants of Alice
Negation: use “!”

Find all descendants of Bob that are not descendants of Alice

Answer

<table>
<thead>
<tr>
<th>x</th>
</tr>
</thead>
<tbody>
<tr>
<td>David</td>
</tr>
</tbody>
</table>
Negation: use “!”

Find all descendants of Bob that are not descendants of Alice

```
// for each person, compute his/her descendants
D(x,y) :- ParentChild(x,y).
D(x,z) :- D(x,y), ParentChild(y,z).
```
Negation: use “!”

Find all descendants of Bob that are not descendants of Alice

```prolog
// for each person, compute his/her descendants
D(x,y) :- ParentChild(x,y).
D(x,z) :- D(x,y), ParentChild(y,z).

// Compute the answer: notice the negation
Q(x) :- D("Bob",x), !D("Alice",x).
```
Same Generation

Two people are in the *same generation* if they are descendants at the same generation of some common ancestor.
Same Generation

Compute pairs of people at the same generation

// common parent
Same Generation

Compute pairs of people at the same generation

// common parent
SG(x,y) :- ParentChild(p,x), ParentChild(p,y)
Same Generation

Compute pairs of people at the same generation

// common parent
SG(x,y) :- ParentChild(p,x), ParentChild(p,y)

// parents at the same generation
Same Generation

Compute pairs of people at the same generation

// common parent
SG(x,y) :- ParentChild(p,x), ParentChild(p,y)

// parents at the same generation
SG(x,y) :- ParentChild(p,x), ParentChild(q,y), SG(p,q)
Same Generation

Compute pairs of people at the same generation

// common parent
SG(x,y) :- ParentChild(p,x), ParentChild(p,y)

// parents at the same generation
SG(x,y) :- ParentChild(p,x), ParentChild(q,y), SG(p,q)

Problem: this includes answers like SG(Carol, Carol)
And also SG(Eve, George), SG(George, Eve)
Same Generation

Compute pairs of people at the same generation

// common parent
SG(x,y) :- ParentChild(p,x), ParentChild(p,y), x < y

// parents at the same generation
SG(x,y) :- ParentChild(p,x), ParentChild(q,y), SG(p,q), x < y
Here are *unsafe* datalog rules. What’s “unsafe” about them?

\[
U_1(x, y) :- \text{ParentChild(“Alice”,} x), \ y \neq “Bob”
\]

\[
U_2(x) :- \text{ParentChild(“Alice”,} x), \neg \text{ParentChild}(x, y)
\]

\[
U_3(\text{minId}, y) :- \text{minId} = \min x : \{ \text{Actor}(x, y, \_ ) \}
\]
Here are *unsafe* datalog rules. What’s “unsafe” about them?

\[
U1(x, y) :- \text{ParentChild(“Alice”,} x), y \neq \text{“Bob”}
\]

\[
U2(x) :- \text{ParentChild(“Alice”,} x), \neg \text{ParentChild}(x, y)
\]

\[
U3(\text{minId, } y) :- \text{minId} = \text{min} x : \{ \text{Actor}(x, y, \_)} \}
\]
Here are *unsafe* datalog rules. What’s “unsafe” about them?

\[
U_1(x,y) \text{ :- ParentChild}(\text{“Alice”},x), \ y \neq \text{“Bob”}
\]

\[
U_2(x) \text{ :- ParentChild}(\text{“Alice”},x), \neg \text{ParentChild}(x,y)
\]

Want Alice’s childless children, but we get all children x (because there exists some y that x is not parent of y)

\[
U_3(\text{minId}, y) \text{ :- } \text{minId} = \min x : \{ \text{Actor}(x, y, _) \}
\]
Here are *unsafe* datalog rules. What’s “unsafe” about them?

\[ U1(x,y) :- \text{ParentChild(“Alice”,}x\text{)}, \ y \neq \text{“Bob”} \]

\[ U2(x) :- \text{ParentChild(“Alice”,}x\text{)}, \neg \text{ParentChild(}x\text{,}y\text{)} \]

Want Alice’s childless children, but we get all children \( x \) (because there exists some \( y \) that \( x \) is not parent of \( y \))

\[ U3(\text{minId, }y) :- \text{minId} = \min x : \{ \text{Actor}(x, y, _) \} \]

Unclear what \( y \) is

Holds for every \( y \) other than “Bob”

\( U1 = \text{infinite!} \)
Here are *unsafe* datalog rules. What’s “unsafe” about them?

U1(x,y) :- ParentChild(“Alice”,x), y != “Bob”

U2(x) :- ParentChild(“Alice”,x), !ParentChild(x,y)

A datalog rule is *safe* if every variable appears in some positive, non-aggregated relational atom.

U3(minId, y) :- minId = min x : { Actor(x, y, _) }
Making Rules Safe

Return pairs \((x,y)\) where \(x\) is a child of Alice, and \(y\) is anybody

\[ U1(x,y) : - ParentChild(“Alice”,x), y != “Bob” \]
Making Rules Safe

Return pairs \((x,y)\) where \(x\) is a child of Alice, and \(y\) is anybody

\[
U_1(x,y) :- \text{ParentChild(“Alice”,}x\text{), } y \neq “Bob”
\]

\[
U_1(x,y) :- \text{ParentChild(“Alice”,}x\text{), Person}(y), y \neq “Bob”
\]
Making Rules Safe

Find Alice’s children who don’t have children.

\[ U_2(x) :\text{ ParentChild(“Alice”,x), \text{ !ParentChild(x,y)} } \]
Making Rules Safe

Find Alice’s children who don’t have children.

\[ U_2(x) : \text{ParentChild}("Alice",x), \neg \text{ParentChild}(x,y) \]

\[ \text{HasChildren}(x) : \text{ParentChild}(x,y) \]
\[ U_2(x) : \text{ParentChild}("Alice",x), \neg \text{HasChildren}(x) \]
Making Rules Safe

Find the smallest Actor ID and his/her first name

\[ \text{U3}(\text{minId}, y) : - \text{minId} = \min x : \{ \text{Actor}(x, y, _) \} \]
Making Rules Safe

Find the smallest Actor ID and his/her first name

\[
U3(\text{minId}, y) :\text{minId} = \min x : \{ \text{Actor}(x, y, _) \}
\]

\[
U3(\text{minId}, y) :\text{minId} = \min x : \{ \text{Actor}(x, _, _) \}, \text{Actor}(\text{minId}, y, _)
\]
Stratified Datalog

- Recursion does not cope well with aggregates or negation
- Example: what does this mean?

\[
\begin{align*}
A() : & \neg B(), \\
B() : & \neg A().
\end{align*}
\]

- A datalog program is *stratified* if it can be partitioned into *strata*
  - Only IDB predicates defined in strata 1, 2, ..., n may appear under \( \neg \) or agg in stratum n+1.

- Many Datalog DBMSs (including souffle) accepts only stratified Datalog.
Stratified Datalog

D(x,y) :- ParentChild(x,y).
D(x,z) :- D(x,y), ParentChild(y,z).
T(p,c) :- D(p,_), c = \text{count} : \{ D(p,y) \}.
Q(d) :- T(p,d), p = “Alice”.

May use D in an agg since it was defined in previous stratum
Stratified Datalog

Stratum 1

\[ D(x,y) : - \text{ParentChild}(x,y). \]
\[ D(x,z) : - D(x,y), \text{ParentChild}(y,z). \]
\[ T(p,c) : - D(p,_), c = \text{count} : \{ D(p,y) \}. \]
\[ Q(d) : - T(p,d), p = \text{“Alice”}. \]

Stratum 2

\[ D(x,y) : - \text{ParentChild}(x,y). \]
\[ D(x,z) : - D(x,y), \text{ParentChild}(y,z). \]
\[ Q(x) : - D(\text{“Alice”},x), \neg D(\text{“Bob”},x). \]

Non-stratified

\[ A() : - \neg B(). \]
\[ B() : - \neg A(). \]

May use \( D \) in an agg since it was defined in previous stratum

May use \( \neg D \)

Cannot use \( \neg A \)
Stratified Datalog

• If we don’t use aggregates or negation, then the Datalog program is already stratified

• If we do use aggregates or negation, it is usually quite natural to write the program in a stratified way
Outline

• Datalog rules
• Recursion
• Semantics
• Negation, aggregates, stratification
• Naïve and Semi-naïve Evaluation
Evaluation

Naïve evaluation: fixpoint semantics:
• At each iteration, compute a relational query
• Repeat until no more change

Semi-naïve evaluation
• Compute only delta’s at each iteration
• Will discuss in another lecture…