CSE544
Data Management
Lectures 15
Datalog
Agenda

• Finish the discussion of datalog

• Brief review of what this class was about
Monotone Queries

- A set function $F(R_1, R_2, ...)$ is monotone if

  $$R_1 \subseteq R_1', R_2 \subseteq R_2', ... \Rightarrow F(R_1, R_2, ...) \subseteq F(R_1', ...)$$
Monotone Queries

- A set function \( F(R_1, R_2, ...) \) is monotone if
  \[
  R_1 \subseteq R_1', R_2 \subseteq R_2', ... \Rightarrow F(R_1, R_2, ...) \subseteq F(R_1', ...)
  \]

- Set difference is not monotone:
  \[
  R_1 - R_2 \not\subseteq R_1 - R_2'
  \]
Monotone Queries

• A set function $F(R_1, R_2, ...) \text{ is monotone if}$

$$R_1 \subseteq R_1', R_2 \subseteq R_2', ... \Rightarrow F(R_1, R_2, ...) \subseteq F(R_1', ...)$$

• Set difference is not monotone:

$$R_1 - R_2 \nsubseteq R_1 - R_2'$$

• Aggregates are not monotone:

$$\{1 + 2\} \nsubseteq \{1 + 2 + 3\}$$
Non-Monotone Features

• Negation

• Aggregates/group-by
Three Useful Queries w/ Negation

Transitive closure of the complement

\[
\begin{align*}
\text{NR}(x, y): & \quad \neg V(x), V(y), \neg R(x, y) \\
\text{T}(x, y): & \quad \neg \text{NR}(x, y) \\
\text{T}(x, y): & \quad \neg \text{NR}(x, z), T(z, y)
\end{align*}
\]
Three Useful Queries w/ Negation

Transitive closure of the complement

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

Complement of the transitive closure

\[
\begin{align*}
\text{T}(x, y): & \neg \text{R}(x, y) \\
\text{T}(x, y): & \neg \text{R}(x, z), \text{T}(z, y) \\
\text{Answ}(x, y): & \neg \text{V}(x), \text{V}(y), \neg \text{T}(x, y)
\end{align*}
\]
Three Useful Queries w/ Negation

Transitive closure of the complement

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

Complement of the transitive closure

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

The Win-Move game (won’t discuss in class)

\[
W(x) : \quad - R(x, y), \neg W(y)
\]
Recursion + Negation Don’t Mix

• The next example is super-simple, but recall a simple fact:

• A relation $A$ of arity 0 is a Boolean variable:
  - $A = \emptyset$ or $A = \{(\ )\}$,
  - i.e. $A$ is either FALSE or TRUE
Recursion + Negation Don’t Mix

\[B(\cdot): \neg \neg A(\cdot)\]
\[A(\cdot): \neg \neg B(\cdot)\]

What are the models?
Recursion+Negation Don’t Mix

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

What are the models? A=False, B=True
Recursion+Negation Don’t Mix

What are the models?

- \( B() : \neg \neg A() \)
- \( A() : \neg \neg B() \)

A=False, B=True
A=True, B=False
Recursion+Negation Don’t Mix

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

What are the models?

A=False, B=True
A= True, B= False
A= True, B= True
Recursion+Negation Don’t Mix

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

What are the models?

A=False, B=True
A=True, B=False
A=True, B=True
No minimal model
Recursion + Negation Don’t Mix

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

What are the models?
- A=False, B=True
- A=True, B=False
- A=True, B=True
- No minimal model

What are the fixpoints?
Recursion + Negation Don’t Mix

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

What are the models?
- A=False, B=True
- A=True, B=False
- A=True, B=True
  No minimal model

What are the fixpoints?
- (False, True), (True, False)
  No least fixpoint
Recursion+Negation Don’t Mix

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

What are the models?
- \(A = \text{False}, B = \text{True}\)
- \(A = \text{True}, B = \text{False}\)
- \(A = \text{True}, B = \text{True}\)
- No minimal model

What are the fixpoints?
- (False,True), (True, False)
- No least fixpoint

What does the naïve algorithm compute?
Recursion+Negation Don’t Mix

\[
B(\cdot) : \neg \neg A(\cdot) \\
A(\cdot) : \neg B(\cdot)
\]

What are the models?

- A=False, B=True
- A=True, B=False
- A=True, B=True
- No minimal model

What are the fixpoints?

(False, True), (True, False)

No least fixpoint

What does the naïve algorithm compute?

\((A_0, B_0) = (0,0)\);
Recursion + Negation Don’t Mix

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

What are the models?
A=False, B=True
A=True, B=False
A=True, B=True
No minimal model

What are the fixpoints?
(False, True), (True, False)

No least fixpoint

What does the naïve algorithm compute?
\((A_0, B_0) = (0,0); (A_1, B_1) = (1,1);\)
Recursion + Negation Don’t Mix

What are the models?
- $A = \text{False}$, $B = \text{True}$
- $A = \text{True}$, $B = \text{False}$
- $A = \text{True}$, $B = \text{True}$
- No minimal model

What are the fixpoints?
- $(\text{False}, \text{True}), (\text{True}, \text{False})$
- No least fixpoint

What does the naïve algorithm compute?
- $(A_0, B_0) = (0,0)$
- $(A_1, B_1) = (1,1)$
- $(A_2, B_2) = (0,0)$
- …
Recursion + Negation Don’t Mix

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

What are the models?
- \( A = \text{False}, B = \text{True} \)
- \( A = \text{True}, B = \text{False} \)
- \( A = \text{True}, B = \text{True} \)
- No minimal model

What are the fixpoints?
- \((\text{False}, \text{True})\), \((\text{True}, \text{False})\)
- No least fixpoint

What does the naïve algorithm compute?
- \((A_0, B_0) = (0,0); (A_1, B_1) = (1,1); (A_2, B_2) = (0,0); \cdots\)
- Does not converge
Approaches to Negation

• Semi-positive datalog

• Stratified datalog

• Sophisticated model-theoretic definitions: stable models, well founded models. Will not discuss.
Semi-positive Datalog

- EDB atoms may be positive or negated
- IDB atoms are positive
- ICO is monotone.

- **Semantics**: least fixpoint of ICO
Semi-positive Datalog

• E.g. transitive closure of complement

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

Intuition:

• Assign IDBs to strata 1, 2, 3, …

• IDBs computed in stratum s, may use non-monotone occurrences of IDBs at strata < s
Stratified Datalog

Formally: assign a stratum $s(R) \in \mathbb{N}$ to each IDB predicate $R$

The program is **stratified** if there exists a stratification such that:
Stratified Datalog

Formally: assign a stratum \( s(R) \in \mathbb{N} \) to each IDB predicate \( R \)

The program is **stratified** if there exists a stratification such that:

- Positive atoms:  
  \[
  \begin{array}{c}
  \text{A}(X): - \cdots \text{B}(Y) \cdots \\
  \end{array}
  \]
  \( s(A) \geq s(B) \)

\[\Box\]
Stratified Datalog

Formally: assign a stratum \( s(R) \in \mathbb{N} \) to each IDB predicate \( R \)

The program is **stratified** if there exists a stratification such that:

- **Positive atoms:** \( A(X): -\cdots B(Y)\cdots \) \( s(A) \geq s(B) \)
- **Negative atoms:** \( A(X): -\cdots \neg B(Y)\cdots \) \( s(A) > s(B) \)
Stratified Datalog

Formally: assign a stratum \( s(R) \in \mathbb{N} \) to each IDB predicate \( R \)

The program is \textit{stratified} if there exists a stratification such that:

- **Positive atoms:** \[ A(X) : \cdots B(Y) \cdots \quad s(A) \geq s(B) \]
- **Negative atoms:** \[ A(X) : \cdots \neg B(Y) \cdots \quad s(A) > s(B) \]
- **Aggregates:** \[ A(\text{agg}(\cdots)) : \neg \text{body} \quad \forall B \in \text{body}: s(A) > s(B) \]
Negation, Aggregates in Souffle

• Negation: !

• Aggregates: complicated syntax, will show by examples
Negation in Souffle

![Graph of family relationships]

<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>
Find all descendants of Bob that are not descendants of Alice.

### Example Table

<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>
Negation in Souffle

Find all descendants of Bob that are not descendants of Alice

Two strata

Dalice(y) :- Child('Alice',y)
Dalice(y) :- Dalice(x), Child(x,y)
Dbob(y) :- Child('Bob',y)
Dbob(y) :- Dbob(x), Child(x,y)
Answ(x) :- Dbob(x), !Dalice(x)
Aggregates in Souffle

Find the minimum id of all Actors called ‘John’
Aggregates in Souffle

Find the minimum id of all Actors called ‘John’

\[
Q(\text{minId}) \; : \; \text{minId} = \text{\texttt{min}} \; x : \{ \text{Actor}(x, y, _), y = \text{\textquoteleft}John\textquoteleft\} \]

Actor(id, fname, lname)
Aggregates in Souffle

Find the minimum id of all Actors called ‘John’

\[
Q(\text{minId}) :- \text{minId} = \text{\texttt{min}} \ x : \{ \text{Actor}(x, y, \_), y = '\text{John}' \}
\]
Aggregates in Souffle

Find the minimum id of all Actors called ‘John’

\[
Q(\text{minId}) \,: \, \text{minId} = \min x : \{ \text{Actor}(x, y, \_), y = 'John' \}
\]
Aggregates in Souffle

Find the minimum id of all Actors called ‘John’

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

Find the minimum id of all Actors called ‘John’

\[ Q(\text{minId}) \leftarrow \text{minId} = \min_x \{ \text{Actor}(x, y, _), y = 'John' \} \]

In SQL

```
SELECT min(id) as minId
FROM Actor as a
WHERE a.fname = 'John'
```
Aggregates in Souffle

- count
- min
- max
- sum
Counting

Count the number of actors called 'John'

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

No variable

Meaning (in SQL, assuming no NULLs)

SELECT count(*) as c
FROM Actor as a
WHERE a.name = 'John'
Group-By

```
Q(y,c) :- Movie(_,_,y), c = count : { Movie(_,_,y) }
```

Meaning (in SQL)

```
SELECT m.year, count(*)
FROM Movie as m
GROUP BY m.year
```
Group-By

For each person, count his/her descendants
Group-By

For each person, count his/her descendants

Alice 4
Bob 5
Carol 3
David 2
Fred 1

Answer
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>

Note: Eve and George do not appear in the answer (why?)
Group-By

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

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

// For each person, count the number of descendants
T(p, c) :- D(p, _), c = count : { D(p, y) }.
// for each person, compute his/her descendants
D(x,y) :- Child(x,y).
D(x,z) :- D(x,y), Child(y,z).

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

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

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

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

Stratified
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
Safe/Unsafe Datalog Rules

• All rules in datalog must be safe

• We have seen only safe rules so far, what is an unsafe rule?

• Examples next, then the definition of safety
Unsafe Datalog Rules

Here are \textit{unsafe} datalog rules. What’s “unsafe” about them?

\begin{verbatim}
U1(x,y) :- Child("Alice",x), y != "Bob"

U2(x) :- Child("Alice",x), !Child(x,y)

U3(minId, y) :- minId = \text{min} x : \{ \text{Actor}(x, y, _) \}
\end{verbatim}
Unsafe Datalog Rules

Here are unsafe datalog rules. What’s “unsafe” about them?

\[
\begin{align*}
U1(x,y) & \leftarrow \text{Child(“Alice”,x)}, \ y \neq \text{“Bob”} \\
U2(x) & \leftarrow \text{Child(“Alice”,x)}, \ !\text{Child(x,y)} \\
U3(\text{minId}, y) & \leftarrow \text{minId} = \min x : \{ \text{Actor}(x, y, _) \}
\end{align*}
\]

y takes infinitely many values
Unsafe Datalog Rules

Here are *unsafe* datalog rules. What’s “unsafe” about them?

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

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

\[ U_3(\text{minId}, y) :- \text{minId} = \text{min } x : \{ \text{Actor}(x, y, _) \} \]

y takes infinitely many values

x has no children?
Or there exists y who is not child of x?
Here are unsafe datalog rules. What’s “unsafe” about them?

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

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

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

Here are *unsafe* datalog rules. What’s “unsafe” about them?

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

U2(x) :- Child(“Alice”,x), !Child(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

\[
U1(x, y) :- \text{Child(“Alice”,}x\text{), } y \neq \text{“Bob”}
\]
Making Rules Safe

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

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

- Unsafe

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

- Safe
Making Rules Safe

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

\[ \text{U2}(x) :\text{- Child(“Alice”,x), !Child(x,y)} \]
Making Rules Safe

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

\[ U_2(x) : \text{Child(“Alice”,x), !Child(x,y)} \]

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

Find the smallest Actor ID and his/her first name

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

Unsafe
Making Rules Safe

Find the smallest Actor ID and his/her first name

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

Unsafe

U3(minId, y) :- minId = min x : { Actor(x, _, _) }, Actor(minID, y, _)

Safe
Recap of the Quarter

• Relational Model:
  – SQL
  – Data Models

• Query Engine:
  – Execution
  – Optimization (3 dimensions)

• Datalog
Some Things We Didn’t Cover

• Transactions
• Provenance
• Tree decomposition, worst-case optimal algorithms
• LSM trees
• Push v.s. pull model
What you should do next

• Finish HW3
• Finish the project, meet on Friday
• Finish the project, present Wednesday
• Finish the project, submit final report
• Submit Review 4
• Finish HW4