CSE 344

JANUARY 29TH – DATALOG
ADMINISTRATIVE MINUTIAE

• HW3 due Friday
• OQ due Wednesday
• HW4 out Wednesday
• Exam next Friday
  • 3:30 - 5:00
WHAT IS DATALOG?

Another query language for relational model

- Designed in the 80’s
- Simple, concise, elegant
- Extends relational queries with recursion

Relies on a logical framework for ”record” selection
**DATALOG: FACTS AND RULES**

Facts = tuples in the database

Rules = queries

**Schema**

- Actor(id, fname, lname)
- Casts(pid, mid)
- Movie(id, name, year)
DATALOG: FACTS AND RULES

Facts = tuples in the database
Rules = queries

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).
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)
DATALOG: FACTS AND RULES

Facts = tuples in the database

Rules = queries

EXTENSIONAL DATABASE PREDICATES = EDB = Actor, Casts, Movie

INTENSIONAL DATABASE PREDICATES = IDB = Q1, Q2, Q3
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

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’.

Book uses AND instead of ,

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

Q(args) :- R1(args) AND R2(args) ....
SEMANTICS OF A SINGLE RULE
Meaning of a datalog rule = a logical statement!

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

- For all x, y, z: if (x,y,z) ∈ Movies and z = ‘1940’ then y is in Q1 (i.e. is part of the answer)
- ∀ x ∀ y ∀ z [(Movie(x,y,z) and z='1940') ⇒ Q1(y)]
- Logically equivalent:
  ∀ y [( ∃ x ∃ z Movie(x,y,z) and z='1940') ⇒ Q1(y)]
- Thus, non-head variables are called "existential variables"
- We want the smallest set Q1 with this property (why?)
DATALOG PROGRAM

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

R encodes a graph

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

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

What does it compute?
R encodes a graph

Initially:
T is empty.

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

T(x,y) :- R(x,y)
T(x,y) :- R(x,z), T(z,y)
Initially:
T is empty.

First iteration:
T =

First rule generates this

Second rule generates nothing
(because T is empty)

What does it compute?
**EXAMPLE**

R encodes a graph

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

Initially:

\[ T = \]

<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 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>

Second iteration:

<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>3</td>
<td>4</td>
</tr>
<tr>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
</tr>
</tbody>
</table>

What does it compute?

New facts

First rule generates this

Second rule generates this
EXAMPLE

R encodes a graph

$R =$

Initially: $T$ is empty.

First iteration:
$T =$

Second iteration:
$T =$

Third iteration:
$T =$

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

What does it compute?
What does it compute?

No new facts. DONE

R encodes a graph

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

Initially: $T$ is empty.

First iteration: $T =$

Second iteration: $T =$

Third iteration: $T =$

Fourth iteration $T =$ (same)
DATALOG SEMANTICS

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:
\[ \emptyset = \text{IDB}_0 \subseteq \text{IDB}_1 \subseteq \text{IDB}_2 \subseteq \ldots \]

It follows that a datalog program w/o functions (+, *, ...) always terminates. (Why? In what time?)
Minimal model semantics:

Return the IDB that

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 smallest IDB satisfying (1)
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
THREE EQUIVALENT
PROGRAMS

R encodes a graph

\begin{itemize}
  \item \[ T(x,y) :- R(x,y) \]
  \item \[ T(x,y) :- R(x,z), T(z,y) \]
  \item \[ T(x,y) :- R(x,y) \]
  \item \[ T(x,y) :- R(x,z), T(z,y) \]
  \item \[ T(x,y) :- T(x,z), R(z,y) \]
  \item \[ T(x,y) :- T(x,z), T(z,y) \]
\end{itemize}

Right linear

Left linear

Non-linear

\begin{array}{|c|c|}
\hline
1 & 2 \\
\hline
2 & 1 \\
\hline
2 & 3 \\
\hline
1 & 4 \\
\hline
3 & 4 \\
\hline
4 & 5 \\
\hline
\end{array}
SAFE DATALOG RULES

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

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

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

Holds for every y other than “Bob”

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

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

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

- **Safe Datalog Rules**
  - Want Alice’s childless children, but we get all children x (because there exists some y that x is not parent of y).
  - **U1** holds for every y other than “Bob.”
    - **U1** = infinite!

ParentChild(p,c)
Here are *unsafe* datalog rules. What’s “unsafe” about them?

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

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

A datalog rule is *safe* if every variable appears in some positive relational atom.
DATALOG: RELATIONAL DATABASE

• Datalog can express things RA cannot
  • Recursive Queries
• Can Datalog express all queries in RA?
DATALOG: RELATIONAL DATABASE

- Datalog can express things RA cannot
  - Recursive Queries
- Can Datalog express all queries in RA?
RELATIONAL ALGEBRA OPERATORS

Union $\cup$, difference $-$
Selection $\sigma$
Projection $\pi$
Cartesian product $\times$, join $\bowtie$
OPERATORS IN DATALOG

• Suppose we want Q1(…) to contain all the values from F1(…) and F2(…)
OPERATORS IN DATALOG

• Suppose we want Q1(...) to contain all the values from F1(...) and F2(...)
  • Q1(...) :- F1(...)
  • Q1(...) :- F2(...)
• What about for difference?
OPERATORS IN DATALOG

• Suppose we want Q1(...) to contain all the values from F1(...) and F2(...)
  • Q1(...) :- F1(...)
  • Q1(...) :- F2(...)

• What about for difference?
  • Q1(...) :- F1(...), !F2(...)
OPERATORS IN DATALOG

• Suppose we want Q1(…) to contain all the values from F1(…) and F2(…)
  • Q1(…) :- F1(…)
  • Q1(…) :- F2(…)

• What about for difference?
  • Q1(…) :- F1(…), !F2(…)
  • The variables (…) in F1 and F2 must be the same, or else we have an unsafe rule
OPERATORS IN DATALOG

• Projection, from the variables $R_1, R_2, \ldots, R_k$
  select some subset of the variables
OPERATORS IN DATALOG

• Projection, from the variables $R_1, R_2, \ldots, R_k$
  select some subset of the variables
  • $Q_1\text{(subset)} : -$ Original(all_attributes)

• Selection: only return certain records from our knowledge base
OPERATORS IN DATALOG

• Projection, from the variables $R_1, R_2, \ldots, R_k$
  select some subset of the variables
  • $Q_1$(subset) :- Original(all_attributes)

• Selection: only return certain records from our knowledge base
  • $Q_1(\ldots)$ :- Original(\ldots), selection_criteria
OPERATORS IN DATALOG

• Cross product: find all the pairs between $R(a_1,a_2\ldots)$ and $S(b_1,b_2\ldots)$
OPERATORS IN DATALOG

• Cross product: find all the pairs between R(a₁,a₂…) and S(b₁,b₂…)
  • Q₁(a₁,b₁,a₂,b₂…) :- R(a₁,a₂…), S(b₁,b₂…)  
• Joins?
  • Natural
OPERATORS IN DATALOG

• Cross product: find all the pairs between \( R(a_1,a_2\ldots) \) and \( S(b_1,b_2\ldots) \)
  
  \[
  Q1(a_1,b_1,a_2,b_2\ldots) \leftarrow R(a_1,a_2\ldots), \ S(b_1,b_2\ldots)
  \]

• Joins?
  
  • Natural: \( Q1(a,b,c) \leftarrow R(a,b), \ S(b,c) \)
  
  • Theta
**OPERATORS IN DATALOG**

- Cross product: find all the pairs between \( R(a_1, a_2 \ldots) \) and \( S(b_1, b_2 \ldots) \)
  - \( Q1(a_1, b_1, a_2, b_2 \ldots) :- R(a_1, a_2 \ldots), S(b_1, b_2 \ldots) \)
- Joins?
  - Natural: \( Q1(a, b, c) :- R(a, b), S(b, c) \)
  - Theta: Cross product with selection
  - Equijoin: subset of Theta join
(SELECT Q.sno
FROM Supplier Q
WHERE Q.sstate = 'WA')
EXCEPT
(SELECT P.sno
FROM Supply P
WHERE P.price > 100)
Datalog:

EXAMPLE

Supplier(sno, sname, scity, sstate)
Part(pno, pname, psize, pcolor)
Supply(sno, pno, price)
EXAMPLE

Datalog:

Q1(no,name,city,state) :-
    Supplier(sno,sname,scity,sstate),
    sstate='WA'
Q2(no,pno,price) :-
    Supply(s,pn,pr),
    pr > 100
Q3(sno) :- Q1(sno,n,c,s)
Q4(sno) :- Q2(sno,pn,pr)
Result(sno) :- Q1(sno),
              !Q2(sno)

Supplier(sno,sname,scity,sstate)
Part(pno,pname,psize,pcolor)
Supply(sno,pno,price)
MORE EXAMPLES W/O RECURSION

Find Joe's friends, and Joe's friends of friends.

\[
\begin{align*}
  A(x) &\text{ :- } \text{Friend('Joe', x)} \\
  A(x) &\text{ :- } \text{Friend('Joe', z), Friend(z, x)}
\end{align*}
\]
MORE EXAMPLES W/O RECURSION

Find all of Joe's friends who do not have any friends except for Joe:

\[
\text{JoeFriends}(x) :- \text{Friend('Joe',x)}
\]
\[
\text{NonAns}(x) :- \text{JoeFriends}(x), \text{Friend}(x,y), y \neq 'Joe'
\]
\[
\text{A}(x) :- \text{JoeFriends}(x), \text{NOT \text{NonAns}(x)}
\]
MORE EXAMPLES W/O RECURSION

Find all people such that all their enemies' enemies are their friends

Q: if someone doesn't have any enemies nor friends, do we want them in the answer?

A: Yes!

\[
\begin{align*}
\text{Everyone}(x) & : \neg \text{Friend}(x,y) \\
\text{Everyone}(x) & : \neg \text{Friend}(y,x) \\
\text{Everyone}(x) & : \neg \text{Enemy}(x,y) \\
\text{Everyone}(x) & : \neg \text{Enemy}(y,x) \\
\text{NonAns}(x) & : \neg \text{Enemy}(x,y), \neg \text{Enemy}(y,z), \neg \text{Friend}(x,z) \\
\text{A}(x) & : \neg \text{Everyone}(x), \neg \text{NonAns}(x)
\end{align*}
\]
Find all persons $x$ that have a friend all of whose enemies are $x$'s enemies.

Everyone($x$) :- Friend($x,y$)
NonAns($x$) :- Friend($x,y$) Enemy($y,z$), NOT Enemy($x,z$)
A($x$) :- Everyone($x$), NOT NonAns($x$)
MORE EXAMPLES W/ RECURSION

Two people are in the same generation if they are siblings, or if they have parents in the same generation.

Find all persons in the same generation with Alice.
MORE EXAMPLES W/ RECURSION

Find all persons in the same generation with Alice
Let’s compute $\text{SG}(x,y) = \text{“}x, y \text{ are in the same generation}\text{”}$

\[
\begin{align*}
\text{SG}(x,y) & \text{ :- ParentChild}(p,x), \text{ ParentChild}(p,y) \\
\text{SG}(x,y) & \text{ :- ParentChild}(p,x), \text{ ParentChild}(q,y), \text{ SG}(p,q) \\
\text{Answer}(x) & \text{ :- SG(“Alice”, x)}
\end{align*}
\]
DATALOG SUMMARY

EDB (base relations) and IDB (derived relations)
Datalog program = set of rules
Datalog is recursive

Some reminders about semantics:

- Multiple atoms in a rule mean join (or intersection)
- Variables with the same name are join variables
- Multiple rules with same head mean union