CSE544
Data Management
Lectures 13
Datalog
Announcement

• Project Milestone due on Monday, 2/26

• HW3 extended to Thursday, 2/29
Project

• Project meetings w/ Dan: Friday, 3/1
• Printing the poster:
  – Kyle can help on Monday, 3/4, OR ask a colleague with a cse account
• Poster presentations: Wed, 3/6, 10-2pm
  – In the atrium of Allen building
  – Setup: 9:30; poster + demo (optional)
  – Snacks, pizza will be provided
Datalog
Motivation

• RA cannot express iteration/recursion
  SQL can, but clumsy, limited

• Data science needs iteration/recursion

• Datalog: designed for recursion
Datalog

- Proposed in the 80’s as “Prolog for DBs”
- Not adopted by industry, no standards
- A darling of academics, hot topic in DB, PL, Networking, …
- In HW4 we will use Souffle
Outline

• Syntax

• Getting familiar with Datalog

• Semantics
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).
Movie(29445, ‘Ave Maria’, 1940).

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(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
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).
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)

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(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 **atom**, or a **relational predicate**

• $R_i(\text{args}_i)$ evaluates to true or false

• Can also have arithmetic predicates, e.g. $z > 1940$
Datalog program

• Datalog program = several rules

• Rules may be recursive!

• Often one IDB is final answer
Outline

• Syntax

• Getting familiar with Datalog

• Semantics
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

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

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

What does it compute?
Example

R encodes a graph

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

Initially:
T is empty.

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

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

What does it compute?
Example

R encodes a graph

Initially: T is empty.

First iteration:
T =

First rule generates this
Second rule generates nothing (because T is empty)

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>

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

\[
T(1,2) \\
T(2,1) \\
T(2,3) \\
T(1,4) \\
T(3,4) \\
T(4,5) \\
\]

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:

\[
T' = \begin{array}{|c|c|}
\hline
1 & 2 \\
1 & 1 \\
2 & 2 \\
1 & 3 \\
2 & 4 \\
1 & 5 \\
3 & 5 \\
\hline
\end{array}
\]

First rule generates this

Second rule generates this

What does it compute?

Example

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

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

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

Initially: \( T \) is empty.

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

First iteration: \( T = \)

Second iteration: \( T = \)

Third iteration: \( T = \)

R encodes a graph

What does it compute?

New fact

Both rules

First rule

Second rule
Example

R encodes a graph

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

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>

Initially: T is empty.

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>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 =

<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>

Iteration k computes pairs (x,y) connected by path of length ≤ k

What does it compute?

No new facts.

DONE
Discussion

• Datalog evaluation is iterative

• It adds new facts at each iteration, stops when nothing new to add

• It always terminates, because the set of possible facts is finite
Example

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

How many iterations until termination?
Example

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

How many iterations until termination?

1 \rightarrow 2 \rightarrow 3 \rightarrow \ldots \rightarrow n
Example

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

How many iterations until termination?

1 2 3 ...

n iterations
Example

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

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

How many iterations until termination?

1 -> 2 -> 3 -> ... -> n

How many iterations until termination?

1 -> 2 -> 3 -> ... -> n
Example

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

How many iterations until termination?

1 → 2 → 3 → ... → n

n iterations

1 → 2 → 3 → ... → n

n iterations

1 → 2 → 3 → ... → n

n iterations
Example

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

How many iterations until termination?

[Diagram showing a sequence of nodes labeled 1, 2, 3, ..., n, with arrows indicating the sequence of iterations. Each iteration involves a step of the form \( T(x,y) \).]
Example

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

How many iterations until termination?

How many iterations on an arbitrary graph G?
Example

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

How many iterations until termination?

1 2 3 ...

n iterations

How many iterations on an arbitrary graph G?

Diameter(G)

37
Three Equivalent Programs

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

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

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

R encodes a graph

<table>
<thead>
<tr>
<th>R=</th>
<th>1</th>
<th>2</th>
<th>4</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>

Right linear

Left linear

Non-linear

How many iterations on an arbitrary graph G?
Three Equivalent Programs

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>

#iterations = diameter
#iterations = log(diameter)

How many iterations on an arbitrary graph G?

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

Right linear

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

Left linear

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

Non-linear

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

Find pairs of nodes \((x,y)\) connected by a path of *even* length

\[ R \]

<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>
Multiple IDBs

R encodes a graph

Find pairs of nodes \((x,y)\) connected by a path of \textit{even} length

<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>

Odd\((x,y)\) :- R\((x,y)\)

Even\((x,y)\) :- Odd\((x,z)\), R\((z,y)\)

Odd\((x,y)\) :- Even\((x,z)\), R\((z,y)\)

Two IDBs: Odd\((x,y)\) and Even\((x,y)\)
Regular Expressions

Labeled Graph:
\[ a(x,y), b(x,y) \]

Find pairs of nodes connected by a path whose labels match
\[ (a.a.b^*)^*.a \]
Labeled Graph: 
\(a(x,y), b(x,y)\)

Find pairs of nodes connected by a path whose labels match 
\((a.a.b^*)^*.a\)

Automaton:
Labeled Graph:
\( a(x,y), b(x,y) \)

Find pairs of nodes connected by a path whose labels match \((a.a.b^*)^*.a\)

Automaton:

\( T_i(x,y) = \text{pairs of nodes connected by paths whose labels match the language accepted by the automaton when the terminal state is } s_i. \)
Labeled Graph:
\( a(x,y), b(x,y) \)

Find pairs of nodes connected by a path whose labels match \((a.a.b^*)^*.a\)

Automaton:

\( T_i(x,y) = \) pairs of nodes connected by a path whose labels match the language accepted by the automaton when the terminal state is \( s_i \).

\( T2(x,y) :- a(x,y) \)
Regular Expressions

Labeled Graph:
\[ a(x,y), b(x,y) \]

Find pairs of nodes connected by a path whose labels match:
\[ (a.a.b^*)^*.a \]

Automaton:

\[ T_i(x,y) = \text{pairs of nodes connected by a paths whose labels match the language accepted by the automaton when the terminal state is } s_i. \]

\[ T2(x,y) ::= a(x,y) \]
\[ T2(x,y) ::= T3(x,z), a(z,y) \]
Regular Expressions

Labeled Graph:
\(a(x,y), b(x,y)\)

Find pairs of nodes connected by a path whose labels match \((a.a.b^*)^*.a\)

Automaton:
\[
\begin{align*}
T_i(x,y) &= \text{pairs of nodes connected by a path whose labels match the language accepted by the automaton when the terminal state is } s_i. \\
T2(x,y) &\ :- \ a(x,y) \\
T2(x,y) &\ :- \ T3(x,z),a(z,y) \\
T3(x,y) &\ :- \ T2(x,z),a(z,y) \\
T3(x,y) &\ :- \ T3(x,z),b(z,y)
\end{align*}
\]
Regular Expressions

Labeled Graph:
\[(a(x,y), b(x,y))\]

Find pairs of nodes connected by a path whose labels match
\[(a.a.b^*)*.a\]

Automaton:

\[T_i(x,y) = \text{pairs of nodes connected by a paths whose labels match the language accepted by the automaton when the terminal state is } s_i.\]

\[T2(x,y) :- a(x,y)\]
\[T2(x,y) :- T3(x,z), a(z,y)\]
\[T3(x,y) :- T2(x,z), a(z,y)\]
\[T3(x,y) :- T3(x,z), b(z,y)\]
\[T4(x,y) :- T3(x,z), a(z,y)\]
Labeled Graph: \(a(x,y), b(x,y)\)

Find pairs of nodes connected by a path whose labels match \((a.a.b^*)^*.a\)

Automaton:

\[T_i(x,y) = \text{pairs of nodes connected by a path whose labels match the language accepted by the automaton when the terminal state is } s_i.\]

\[\begin{align*}
T2(x,y) & := a(x,y) \\
T2(x,y) & := T3(x,z), a(z,y) \\
T3(x,y) & := T2(x,z), a(z,y) \\
T3(x,y) & := T3(x,z), b(z,y) \\
T4(x,y) & := T3(x,z), a(z,y) \\
\text{Answ}(x,y) & := T4(x,y)
\end{align*}\]
Recursion in SQL

• SQL supports a limited form of recursion by using Common Table Expression (CTE)
Recursion in SQL

with recursive T as
    (select * from R
     union
     select distinct R.x as x, T.y as y from R, T
     where R.y=T.x)
select * from T;

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

T is called a CTE
Recursion in SQL

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

with recursive T as

\[
\begin{align*}
\text{(select * from R} \\
\text{union}
\text{select distinct R.x as x, T.y as y from R, T} \\
\text{where R.y=T.x})
\end{align*}
\]

select * from T;

If you forgot ‘distinct’, then it diverges
Recursion in SQL

Clumsy, restricted, inefficient:

• Only a single IDB
• Only linear query
• Only this structure:
  – (non-recursive) union (recursive)
• Set or bag semantics (which diverges)
Outline

• Syntax

• Getting familiar with Datalog

• Semantics
Semantids of Datalog

Datalog has three equivalent ways to define its semantics. We consider two:

• Least fixpoint semantics

• Minimal model semantics
Immediate Consequence Operator

- The Immediate Consequence Operator (ICO) is a query that takes all EDBs, all IDBs, and computes a new state of the IDBs, by applying all rules.
Immediate Consequence Operator

• The Immediate Consequence Operator (ICO) is a query that takes all EDBs, all IDBs, and computes a new state of the IDBs, by applying all rules

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

\[
\text{ICO} \\
R(x, y) \cup \Pi_{xy}(R(x, z) \bowtie T(z, y))
\]
Immediate Consequence Operator

• A function $f$ is monotone if:

$$R_1 \subseteq R'_1, R_2 \subseteq R'_2, \ldots :$$

$$f(R_1, R_2, \ldots) \subseteq f(R'_1, R'_2, \ldots)$$
Immediate Consequence Operator

• A function $f$ is monotone if:

\[
R_1 \subseteq R_1', R_2 \subseteq R_2', \ldots : \\
f(R_1, R_2, \ldots) \subseteq f(R_1', R_2', \ldots)
\]

• The ICO is a monotone function, because it uses only $\bowtie, \Pi, \sigma, \cup$

• The only non-monotone operator is -
1. Fixpoint Semantics

• x is a fixpoint of a function f if f(x) = x
1. Fixpoint Semantics

- $x$ is a fixpoint of a function $f$ if $f(x) = x$
- $x$ is the least fixpoint if for any other fixpoint $y$, it holds that $x \subseteq y$
1. Fixpoint Semantics

- x is a fixpoint of a function f if $f(x) = x$
- x is the least fixpoint if for any other fixpoint y, it holds that $x \subseteq y$

**Definition.** The semantics of a datalog program is the least fixpoint of the ICO.
1. Fixpoint Semantics

• x is a fixpoint of a function f if f(x)=x
• x is the least fixpoint if for any other fixpoint y, it holds that x ⊆ y

• **Definition.** The semantics of a datalog program is the least fixpoint of the ICO

• Next: we prove that it exists.
1. Fixpoint Semantics

Naïve evaluation algorithm

Start: IDB₀ = ∅;  t = 0
Repeat:
  IDB_{t+1} = ICO(EDB, IDBₜ)
  t = t+1
Until IDBₜ = IDB_{t-1}
1. Fixpoint Semantics

Naïve evaluation algorithm

Start: $\text{IDB}_0 = \emptyset; \quad t = 0$
Repeat:
    \[ \text{IDB}_{t+1} = \text{ICO(EDB, IDB}_t) \]
    \[ t = t + 1 \]
Until $\text{IDB}_t = \text{IDB}_{t-1}$

Fact: $\emptyset = \text{IDB}_0 \subseteq \text{IDB}_1 \subseteq \text{IDB}_2 \subseteq ...$
1. Fixpoint Semantics

Naïve evaluation algorithm

Start: $\text{IDB}_0 = \emptyset$; $t = 0$
Repeat:

$\text{IDB}_{t+1} = \text{ICO(EDB, IDB}_t)$
$t = t+1$

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

Fact: $\emptyset = \text{IDB}_0 \subseteq \text{IDB}_1 \subseteq \text{IDB}_2 \subseteq \ldots$
Proof by induction. $\emptyset = \text{IDB}_0 \subseteq \text{IDB}_1$
1. Fixpoint Semantics

Naïve evaluation algorithm

Start: \( IDB_0 = \emptyset; \ t = 0 \)

Repeat:

\[
IDB_{t+1} = ICO(EDB, IDB_t)
\]

\[
t = t+1
\]

Until \( IDB_t = IDB_{t-1} \)

**Fact:** \( \emptyset = IDB_0 \subseteq IDB_1 \subseteq IDB_2 \subseteq ... \)

**Proof** by induction. \( \emptyset = IDB_0 \subseteq IDB_1 \)

If \( IDB_{t-1} \subseteq IDB_t \)

then \( IDB_t = ICO(IDB_{t-1}) \subseteq ICO(IDB_t) = IDB_{t+1} \)
1. Fixpoint Semantics

Naïve evaluation algorithm

Start: $\text{IDB}_0 = \emptyset$; $t = 0$
Repeat:

$\text{IDB}_{t+1} = \text{ICO}(\text{EDB}, \text{IDB}_t)$
$t = t + 1$

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

Fact: $\emptyset = \text{IDB}_0 \subseteq \text{IDB}_1 \subseteq \text{IDB}_2 \subseteq \ldots$
1. Fixpoint Semantics

Naïve evaluation algorithm

Start: $\text{IDB}_0 = \emptyset; \ t = 0$
Repeat:
$$\text{IDB}_{t+1} = \text{ICO}(\text{EDB}, \text{IDB}_t)$$
$$t = t + 1$$
Until $\text{IDB}_t = \text{IDB}_{t-1}$

Fact: $\emptyset = \text{IDB}_0 \subseteq \text{IDB}_1 \subseteq \text{IDB}_2 \subseteq \ldots$
Fact: There exists $t_0$ such that $\text{IDB}_{t_0} = \text{IDB}_{t_0+1}$

Fixpoint!
1. Fixpoint Semantics

Naïve evaluation algorithm

Start: $\text{IDB}_0 = \emptyset$; $t = 0$
Repeat:

$\text{IDB}_{t+1} = \text{ICO}(\text{EDB}, \text{IDB}_t)$
$t = t+1$

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

**Fact:** $\emptyset = \text{IDB}_0 \subseteq \text{IDB}_1 \subseteq \text{IDB}_2 \subseteq \ldots$

**Fact:** There exists $t_0$ such that $\text{IDB}_{t_0} = \text{IDB}_{t_0+1}$  

**Proof.** Because the number of possible tuples from EDBs is finite.
1. Fixpoint Semantics

Naïve evaluation algorithm

Start: $\text{IDB}_0 = \emptyset; \ t = 0$
Repeat:

$\text{IDB}_{t+1} = \text{ICO}(\text{EDB}, \text{IDB}_t)$
$t = t + 1$
Until $\text{IDB}_t = \text{IDB}_{t-1}$

Fact: $\emptyset = \text{IDB}_0 \subseteq \text{IDB}_1 \subseteq \text{IDB}_2 \subseteq \ldots$
Fact: There exists $t_0$ such that $\text{IDB}_{t_0} = \text{IDB}_{t_0+1}$

Fixpoint!
1. Fixpoint Semantics

Naïve evaluation algorithm

Start: $\text{IDB}_0 = \emptyset; \ t = 0$
Repeat:

$\text{IDB}_{t+1} = \text{ICO}(\text{EDB}, \text{IDB}_t)$
$t = t+1$
Until $\text{IDB}_t = \text{IDB}_{t-1}$

**Fact:** $\emptyset = \text{IDB}_0 \subseteq \text{IDB}_1 \subseteq \text{IDB}_2 \subseteq ...$

**Fact:** There exists $t_0$ such that $\text{IDB}_{t_0} = \text{IDB}_{t_0+1}$

**Fact:** if IDB is any fixpoint, then $\forall t, \text{IDB}_t \subseteq \text{IDB}$

Fixpoint!
1. Fixpoint Semantics

Naïve evaluation algorithm

Start: \( IDB_0 = \emptyset; \ t = 0 \)
Repeat:
\[
IDB_{t+1} = ICO(EDB, IDB_t) \\
t = t + 1
\]
Until \( IDB_t = IDB_{t-1} \)

Fact: \( \emptyset = IDB_0 \subseteq IDB_1 \subseteq IDB_2 \subseteq \ldots \)
Fact: There exists \( t_0 \) such that \( IDB_{t_0} = IDB_{t_0+1} \)
Fact: if \( IDB \) is any fixpoint, then \( \forall t, IDB_t \subseteq IDB \)
Proof. Induction on \( t \). \( \emptyset = IDB_0 \subseteq IDB \)
1. Fixpoint Semantics

Naïve evaluation algorithm

Start: $\text{IDB}_0 = \emptyset$; $t = 0$
Repeat:
\[
\text{IDB}_{t+1} = \text{ICO}(\text{EDB}, \text{IDB}_t) \\
t = t + 1
\]
Until $\text{IDB}_t = \text{IDB}_{t-1}$

**Fact:** $\emptyset = \text{IDB}_0 \subseteq \text{IDB}_1 \subseteq \text{IDB}_2 \subseteq \ldots$

**Fact:** There exists $t_0$ such that $\text{IDB}_{t_0} = \text{IDB}_{t_0+1}$

**Fact:** if IDB is any fixpoint, then $\forall t, \text{IDB}_t \subseteq \text{IDB}$

**Proof.** Induction on $t$. $\emptyset = \text{IDB}_0 \subseteq \text{IDB}$
If $\text{IDB}_t \subseteq \text{IDB}$ then $\text{IDB}_{t+1} = \text{ICO}(\text{IDB}_t) \subseteq \text{ICO}(\text{IDB}) = \text{IDB}$
1. Fixpoint Semantics

Naïve evaluation algorithm

Start: \( \text{IDB}_0 = \emptyset; \ t = 0 \)
Repeat:
\[
\begin{align*}
\text{IDB}_{t+1} &= \text{ICO}(\text{EDB}, \text{IDB}_t) \\
t &= t+1
\end{align*}
\]
Until \( \text{IDB}_t = \text{IDB}_{t-1} \)

Fact: \( \emptyset = \text{IDB}_0 \subseteq \text{IDB}_1 \subseteq \text{IDB}_2 \subseteq \ldots \)
Fact: There exists \( t_0 \) such that \( \text{IDB}_{t_0} = \text{IDB}_{t_0+1} \) \hspace{1cm} \text{Fixpoint!}
Fact: if \( \text{IDB} \) is any fixpoint, then \( \forall t, \text{IDB}_t \subseteq \text{IDB} \)

Corollary. The Least Fixpoint of the ICO exists, and is computed by the Naïve Algorithm
Datalog and Logic

We need:

• A Quick review of Boolean Logic, FO
• Datalog as logical sentences
Boolean Logic

- Propositional symbols: p, q, r, ...
- Boolean connectives: ∨, ∧, ¬, ⇒
- (p ∨ q) ∧ (q ∨ ¬r) ∧ ¬(p ∧ q ∨ r)
Boolean Logic

• Propositional symbols: p, q, r, …
• Boolean connectives: ∨, ∧, ¬, ⇒
• \((p \lor q) \land (q \lor \neg r) \land \neg(p \land q \lor r)\)
• Things to know:
  – De Morgan: \(\neg(p \lor q) = \neg p \land \neg q\) and dual
  – Implications: \(p \Rightarrow q \equiv \neg p \lor q\)
  – Therefore: \(\neg(p \Rightarrow q) \equiv p \land \neg q\)
First Order Logic

- Relation symbols, variables, ops $\lor, \land, \neg, \Rightarrow, \forall, \exists$
- A **sentence** is a formula w/o free vars
- A **model** is a database that makes the formula true
First Order Logic

- Relation symbols, variables, ops $\lor, \land, \neg, \Rightarrow, \forall, \exists$
- A **sentence** is a formula w/o free vars
- A **model** is a database that makes the formula true
- What are the models of:
  - $\exists x \exists y \exists z (R(x, y) \land R(y, z))$
  - $\exists x \forall y (R(x, y))$
First Order Logic

• Relation symbols, variables, ops $\lor, \land, \neg, \Rightarrow, \forall, \exists$
• A sentence is a formula w/o free vars
• A model is a database that makes the formula true
• What are the models of:
  – $\exists x \exists y \exists z (R(x, y) \land R(y, z))$
  – $\exists x \forall y (R(x, y))$
• Things to know:
  – De Morgan $\neg \forall x (...) \equiv \exists x \neg (...)$
First Order Logic

• Relation symbols, variables, ops $\lor, \land, \neg, \Rightarrow, \forall, \exists$
• A **sentence** is a formula w/o free vars
• A **model** is a database that makes the formula true
• What are the models of:
  - $\exists x \exists y \exists z (R(x, y) \land R(y, z))$
  - $\exists x \forall y (R(x, y))$
• Things to know:
  - De Morgan $\neg \forall x (\ldots) \equiv \exists x \neg (\ldots)$
  - $\forall x \forall y (R(x, y) \Rightarrow T(x)) \equiv \forall x (\exists y R(x, y) \Rightarrow T(x))$
First Order Logic

• Relation symbols, variables, ops $\lor, \land, \neg, \Rightarrow, \forall, \exists$

• A **sentence** is a formula w/o free vars

• A **model** is a database that makes the formula true

• What are the models of:
  - $\exists x \exists y \exists z (R(x, y) \land R(y, z))$
  - $\exists x \forall y (R(x, y))$

• Things to know:
  - De Morgan $\neg \forall x (\ldots) \equiv \exists x \neg (\ldots)$
  - $\forall x \forall y (R(x, y) \Rightarrow T(x)) \equiv \forall x (\exists y R(x, y) \Rightarrow T(x))$
    Because $\forall x \forall y (\neg R(x, y) \lor T(x)) \equiv \forall x ((\forall y \neg R(x, y)) \lor T(x))$
A datalog rule is a Sentence

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

This is why a non-head variable is called "existential" variable

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

\[
\forall y \ [(\exists x \exists z \text{ Movie}(x,y,z) \text{ and } z='1940') \Rightarrow Q1(y)]
\]
2. Minimal Model Semantics:

• Let $\Phi_P$ be the sentence that is the conjunction of all rules of the datalog program $P$
• A model of $P$ is an IDB instance that is a model of $\Phi_P$
• The minimal model of $P$ is a model that is contained in all other models
2. Minimal Model Semantics:

- **Definition.** The minimal model semantics of a program P is the minimal model of P.

- **Theorem.** The minimal model exists and coincides with the least fixpoint of P.
Example

R encodes a graph

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

R =

<table>
<thead>
<tr>
<th></th>
<th>2</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>
</tbody>
</table>
Example

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

1. Least fixpoint semantics:
Repeat \( T_{t+1}(x,y) := R(x,y) \cup \Pi_{xy}(R(x,z) \bowtie T(z,y)) \)
Example

R encodes a graph

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

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

1. Least fixpoint semantics:
Repeat \( T_{t+1}(x,y) := R(x,y) \cup \Pi_{xy}(R(x,z) \bowtie T(z,y)) \)

2. Minimal model semantics
which one is a model? A minimal model?

\[
\begin{array}{ccc}
2 & 1 & 2 \\
2 & 1 & 3 \\
2 & 3 & \ldots \\
\end{array}
\]

\[
\begin{array}{ccc}
1 & 2 \\
2 & 1 \\
2 & 1 \\
1 & 3 \\
2 & 2 \\
1 & 3 \\
2 & 2 \\
\end{array}
\]

\[
\begin{array}{ccc}
1 & 1 \\
1 & 2 \\
1 & 3 \\
\ldots & \ldots \\
3 & 1 \\
2 & 2 \\
3 & 2 \\
3 & 3 \\
\end{array}
\]
Example

$$R = \begin{array}{cc}
1 & 2 \\
2 & 1 \\
2 & 3 \\
\end{array}$$

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

1. Least fixpoint semantics:
Repeat $T_{t+1}(x, y) := R(x, y) \cup \Pi_{xy}(R(x, z) \bowtie T(z, y))$

2. Minimal model semantics
which one is a model? A minimal model?

This is the minimal model
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

• Analogous to SQL and RA

Next week: aggregates, negation, semi-naïve evaluation