Datalog Program

• **Logical Programming:**
  - Finding solution to a set of requirements given as logical rules

• **Program example:**

  \[ \text{married\_man}(Y) \leftarrow \text{married\_to}(X, Y). \]

  - **Input:**
    - Woman: Sue, Bob
    - Man: Ed
    - Married to: Sue, Bob

  - **Output:**
    - Married Man: Bob

Basic Definitions

• An **atomic formula** has the form \( R(t_1, \ldots, t_k) \) where:
  - \( R \) is a \( k \)-ary relation symbol
  - Each \( t_i \) is either a constant or a variable

• A **Datalog rule** has the form

  \[ \text{head} \leftarrow \text{body} \]

  where **head** is an atomic formula and **body** is a sequence of atomic formulas

  - For simplicity, we disallow constants in the head

• A **Datalog program** is a finite set of Datalog rules

EDBs and IDBs

• **Datalog rules** operates over:

  - **Extensional Database (EDB) predicates**
    - These are the provided/stored database relations from the relational schema

  - **Intentional Database (IDB) predicates**
    - These are the relations derived from the stored relations through the rules
    - Each IDB appears as a head of some rule

  \[ \text{married\_man}(Y) \leftarrow \text{married\_to}(X, Y). \]
Logical Interpretation of a Rule

- The rule
  \[ \text{married\_man}(Y) \leftarrow \text{married\_to}(X, Y). \]
  is interpreted as the logical rule:
  \[ \forall Y \exists X [\text{married\_to}(X, Y) \rightarrow \text{married\_man}(Y)] \]
  which is equivalent to
  \[ \forall Y \forall X [\text{married\_to}(X, Y) \rightarrow \text{married\_man}(Y)] \]

Logical Interpretation of a Rule

- The rule
  \[ \text{married\_man}(Y) \leftarrow \text{married\_to}(X, Y), \text{man}(Y). \]
  is interpreted as the logical rule:
  \[ \forall Y \exists X [\text{married\_to}(X, Y) \land \text{man}(Y) \rightarrow \text{married\_man}(Y)] \]
  which is equivalent to
  \[ \forall Y \forall X [\text{married\_to}(X, Y) \land \text{man}(Y) \rightarrow \text{married\_man}(Y)] \]

Semantics of Datalog Programs

- Datalog programs \( P \) are defined over a schema
  - This schema contains EDB+IDB relation symbols
  - The input to \( P \) is an instance \( I \) over the EDB schema
  - The output of \( P \) is an instance \( J \) over the IDB schema

Model-Theoretic Definition

- We say that \( J \) is a model of \( P \) (w.r.t. \( I \)) if \( I \cup J \) satisfies all the rules of \( P \)
- We say that \( J \) is a minimal model is \( J \) does not properly contain any other model
  \[ \text{married\_man}(Y) \leftarrow \text{married\_to}(X, Y). \]

Model-Theoretic Definition

- The following is also a model for \( P \)
  - The logical rule evaluates to true on \( I \)
    \[ \forall Y \forall X [\text{married\_to}(X, Y) \rightarrow \text{married\_man}(Y)] \]
  - However, this model is not minimal
    - We can omit a tuple and still remain with a model

Outline

- Datalog programs
  - Basic definitions
  - EDBs and IDBs
- Semantics
  - Logical interpretation
  - Model theoretic semantics
- Safety
- Extensions
  - Recursion
  - Negation
- Questions
Safety in Datalog

- What is the problem with the Datalog rule $q(X,Y) \leftarrow p(X)$?
- Our goal:
  - Finite output
  - Independent of the domain
- A safe rule is a rule in which
  - Every variable $x$ is bounded, i.e., it appears in an atom $R(\ldots,x,\ldots)$ in the body of some rule

Outline

- Datalog programs
  - Basic definitions
  - EDBs and IDBs
- Semantics
  - Logical interpretation
  - Model theoretic semantics
- Safety
  - Extensions
    - Recursion
    - Negation
      - stratification
- Questions

Recursive Datalog

- Let us consider the following Datalog program:
  
  \[
  \begin{align*}
  \text{Ancestor}(A,D) & \leftarrow \text{Father}(A,D) \\
  \text{Ancestor}(A,D) & \leftarrow \text{Ancestor}(A,P), \text{Father}(P,D)
  \end{align*}
  \]
- This is a recursive program
  - Ancestor is defined in terms of itself
  - Can a non-recursive program compute Ancestor?

Recursive Datalog

- The dependency graph of a Datalog program is the directed graph $(V,E)$ where
  - $V$ is the set of IDB predicates (relation names)
  - $E$ contains an edge $R \rightarrow S$ whenever there is a rule with $S$ in the head and $R$ in the body
- A Datalog program is recursive if its dependency graph contains a cycle
- With recursion we can express transitive closure
  - Cannot be done without recursion

Datalog with negation

- Input:
  
  \[
  \begin{array}{c|c|c|c|}
  \text{Woman} & \text{Man} & \text{Married to} \\
  \hline
  \text{Sue} & \text{Bob} & \text{Sue} \ \text{Bob} \\
  \text{Ed} & & \\
  \end{array}
  \]
- There are two possible minimal models:
  - The first includes $\text{married_man}(\text{Bob})$ and $\text{bachelor}(\text{Ed})$
  - The second includes $\text{married_man}(\text{Bob})$ and $\text{bachelor}(\text{Ed})$

Stratified Programs

- We need to change the semantics definition when we have negation
  - Intuitively, we want to first fully evaluate the relation $\text{married_man}$ and then move to compute the relation $\text{bachelor}$
- We define the semantics by defining stratification
  - Partitioning the IDB relations to “layers”
Stratified Programs

- Let \( P \) be a Datalog program
- Let \( E_0 \) be set of EDB predicates
- A stratification of \( P \) is a partitioning of the IDBs into disjoint sets \( E_1, \ldots, E_k \) where:
  - For \( i = 1, \ldots, k \), every rule with head in \( E_i \) has body predicates only from \( E_0, \ldots, E_{i-1} \)
  - For \( i = 1, \ldots, k \), every rule with head in \( E_i \) can have negated body predicates only from \( E_0, \ldots, E_{i-1} \)
- In general there might be more than one stratification!
- Note that all of them will lead to the same semantics.

### Stratified Programs - example

\[
\begin{align*}
\text{married\_man}(Y) & \leftarrow \text{married\_to}(X, Y). \\
\text{bachelor}(Y) & \leftarrow \text{man}(Y), \neg \text{married\_man}(Y)
\end{align*}
\]

- In our case
  - \( E_0 \) includes the relation symbol \( \text{married\_to}, \text{man} \)
  - \( E_1 - \text{married\_man} \)
  - \( E_2 - \text{bachelor} \)

### Datalog with negation

\[
\begin{align*}
\text{married\_man}(Y) & \leftarrow \text{married\_to}(X, Y). \\
\text{bachelor}(Y) & \leftarrow \text{man}(Y), \neg \text{married\_man}(Y)
\end{align*}
\]

- The evaluation
  - \( E_0 \)
    - \text{Woman}: Sue, Ed
    - \text{Man}: Bob, Sue
    - \text{Married to}: Sue, Bob
  - \( E_1 \)
    - \text{Married\_man}: Bob
  - \( E_2 \)
    - \text{bachelor}: Ed

### Negation and safety

- Reminder:
  - A safe rule is a rule in which
    - Every variable \( x \) is bounded, i.e., it appears in an atom \( R(\ldots,x,\ldots) \) in the body of some rule
  - Appearing in a negated atom does not bound the variable
    - The following rule is not safe
      \[
      \text{bachelor}(Y) \leftarrow \neg \text{married\_man}(Y)
      \]
    - To make it safe we must bound \( Y \), i.e.,
      \[
      \text{bachelor}(Y) \leftarrow \neg \text{married\_man}(Y), \text{man}(Y)
      \]

### Examples

- Write a Datalog program that defines the Binary relation \( \text{never\_married}(x,y) \) where \( x \) is a woman that is not married to \( y \)

  - 1st try:
    \[
    \text{never\_married}(x,y) \leftarrow \neg \text{married\_to}(x,y)
    \]
    Incorrect!
  - 2nd try:
    \[
    \text{never\_married}(x,y) \leftarrow \text{man}(y), \text{woman}(x), \neg \text{married\_to}(x,y)
    \]

### Outline

- Datalog programs
  - Basic definitions
  - EDBs and IDBs
- Semantics
  - Logical interpretation
  - Model theoretic semantics
- Safety
- Extensions
  - Recursion
  - Negation
- Questions
Question from Exam

- Assume we have the following database
  - Event(place, time)
  - Person(id)
  - Seen(id, place, time)
- A social path between persons p and p' is a sequence p₁, p₂, ..., pₙ such that
  - p = p₁ and p' = pₙ
  - For every i there exists an event such that both pᵢ, pᵢ₊₁ have participated in
- Write a Datalog program (possibly with negation) that defines the relation Out(i,i') such that
  - there exists a social path between i and i'
  - i and i' haven’t participated in the same event

Answer

TogetherEvent(I, I') ← Person(I), Person(I'), Event(p,t), Seen(I,p,t), Seen(I',p,t)

SocialPath(I, I') ← TogetherEvent(I, I')
SocialPath(I, I') ← SocialPath(I, J), TogetherEvent(J, I')

Out(I,I') ← SocialPath(I,I'), ~TogetherEvent(I, I')