Lecture 5: Queries in Logic
Mathematical logic has had an immense impact on CS
Computing has strongly driven one particular branch of logic: finite model theory
- That is, (FO/SO) logic restricted to finite models
- Very strong connections to complexity theory
- The basis of branches in Artificial Intelligence
It is a natural tool to capture and attack fundamental problems in database management
- Relations as first-class citizens
- Inference for assuring data integrity
  - Inference for question answering (queries)
It has been used for developing and analyzing the relational model from the early days (Codd, 1972)
Outline

• Introduction

• Relational Calculus
  ▪ Syntax and Semantics
  ▪ Domain Independence and Safety
  ▪ Equivalence to RA

• Datalog
  ▪ Syntax and Semantics
  ▪ Recursion
  ▪ Negation
Relational Calculus (RC)

• RC is, essentially, first-order logic (FO) over the schema relations
  – A query has the form “find all tuples \((x_1, \ldots, x_k)\) that satisfy an FO condition”

• RC is a *declarative* query language
  – Meaning: a query is not defined by a sequence of operations, but rather by a condition that the result should satisfy
Outline

• Introduction
• Relational Calculus
  ▪ Syntax and Semantics
  ▪ Domain Independence and Safety
  ▪ Equivalence to RA
• Datalog
  ▪ Syntax and Semantics
  ▪ Recursion
  ▪ Negation
Which relatives does this query find?

Answer: Canadian aunts
• Constant values: a, b, c, ...
  – Values that may appear in table cells

• Variables: x, y, z, ...
  – Range over the values that may appear in table cells

• Relation symbols: R, S, T, Person, Parent, ...
  – Each with a specified arity
  – Will be fixed by the relational schema at hand
  – No attribute names, only attribute positions!
Atomic RC Formulas

- Atomic formulas:
  - $R(t_1, \ldots, t_k)$
    - $R$ is a $k$-ary relation
    - Each $t_i$ is a variable or a constant
    - Semantically it states that $(t_1, \ldots, t_k)$ is a tuple in $R$
    - Example: $\text{Person}(x, \text{'female'}, \text{'Canada'})$
  - $x \, \text{op} \, u$
    - $x$ is a variable, $u$ is a variable/constant, $\text{op}$ is one of $>, <, =, \neq$
    - Example: $x=y$, $z>5$
RC Formulas

- Formula:
  - Atomic formula
  - If $\phi$ and $\psi$ are formulas then these are formulas:
    
    $\phi \land \psi \quad \phi \lor \psi \quad \phi \rightarrow \psi \quad \neg \phi \quad \exists x \, \phi \quad \forall x \, \phi$

    
    Person($u$, 'female', 'Canada') $\land$

    $\exists y, z [Parent(y, x) \land Parent(z, y) \land$

    $\exists w [Parent(z, w) \land y \neq w \land (u = w \lor Spouse(u, w))]$
Free Variables

• Informally, variables not bound to quantifiers

• Formally:
  – A free variable of an atomic formula is a variable that occurs in the atomic formula
  – A free variable of $\varphi \land \psi$, $\varphi \lor \psi$, $\varphi \rightarrow \psi$ is a free variable of either $\varphi$ or $\psi$
  – A free variable of $\neg \varphi$ is a free variable of $\varphi$
  – A free variable of $\exists x \varphi$ and $\forall x \varphi$ is a free variable $y$ of $\varphi$ such that $y \neq x$

• The notation $\varphi(x_1,\ldots,x_k)$ implies that $x_1,\ldots,x_k$ are the free variables of $\varphi$ (in some order)
What Are the Free Variables?

\[
\begin{align*}
\text{Person}(u, 'female', 'Canada') \land \\
\exists y,z [ \text{Parent}(y,x) \land \text{Parent}(z,y) \land \\
\exists w [ \text{Parent}(z,w) \land y \neq w \land (u = w \lor \text{Spouse}(u,w))] ]
\end{align*}
\]

\[\varphi(x,u), \text{CandianAunt}(u,x), \ldots\]

\[
\begin{align*}
\text{Person}(u, 'female', v) \land \\
\exists y,z [ \text{Parent}(y,x) \land \text{Parent}(z,y) \land \\
\text{Parent}(z,w) \land y \neq w \land (u = w \lor \text{Spouse}(u,w)) ]
\end{align*}
\]
• An **RC query** is an expression of the form

\[ \{ (x_1, \ldots, x_k) \mid \varphi(x_1, \ldots, x_k) \} \]

where \( \varphi(x_1, \ldots, x_k) \) is an RC formula

• An RC query is **over** a relational schema \( s \) if all the relation symbols belong to \( s \) (with matching arities)
\[ \{ (x,u) \mid \text{Person}(u, \text{'female'}, \text{'Canada'}) \land \\
\exists y,z[\text{Parent}(y,x) \land \text{Parent}(z,y) \land \\
\exists w [\text{Parent}(z,w) \land y \neq w \land (u = w \lor \text{Spouse}(u,w))] \} \]
### Studies

<table>
<thead>
<tr>
<th>sid</th>
<th>student</th>
<th>course</th>
</tr>
</thead>
<tbody>
<tr>
<td>861</td>
<td>Alma</td>
<td>DB</td>
</tr>
<tr>
<td>861</td>
<td>Alma</td>
<td>PL</td>
</tr>
<tr>
<td>753</td>
<td>Amir</td>
<td>DB</td>
</tr>
<tr>
<td>753</td>
<td>Amir</td>
<td>AI</td>
</tr>
<tr>
<td>955</td>
<td>Ahuva</td>
<td>PL</td>
</tr>
<tr>
<td>955</td>
<td>Ahuva</td>
<td>DB</td>
</tr>
<tr>
<td>955</td>
<td>Ahuva</td>
<td>AI</td>
</tr>
</tbody>
</table>

### CourseType

<table>
<thead>
<tr>
<th>course</th>
<th>type</th>
</tr>
</thead>
<tbody>
<tr>
<td>DB</td>
<td>core</td>
</tr>
<tr>
<td>PL</td>
<td>core</td>
</tr>
<tr>
<td>AI</td>
<td>elective</td>
</tr>
<tr>
<td>DC</td>
<td>elective</td>
</tr>
</tbody>
</table>

**Who took all core courses?**

\[
\text{Studies} \div \pi_{\text{course}} \sigma_{\text{type}=\text{core}} \text{CourseType}
\]
R \div S in Primitive RA vs. RC

\[ R(X,Y) \div S(Y) \]

In RA:
\[ \pi_X R \setminus \pi_X \left( (\pi_X R \times S) \setminus R \right) \]

In RC:
\[ \{ (X) \mid \exists Z [R(X,Z)] \land \forall Y [S(Y) \rightarrow R(X,Y)] \} \]
DRC vs. TRC

- There are two common variants of RC:
  - **DRC**: *Domain Relational Calculus* (what we're doing)
  - **TRC**: *Tuple Relational Calculus*

- DRC applies vanilla FO: variables span over *attribute values*, relations have *arity* but no attribute names

- TRC is more database friendly: variables span over *tuples*, where a tuple has named attributes

- There are easy conversions between the two formalisms; nothing deep
Our Example in TRC

- Person(id, gender, country)
- Parent(parent, child)
- Spouse(person1, person2)

\[
\{ t \mid \exists p \ [ p \in \text{Person} \land p[\text{gender}] = 'female' \land p[\text{country}] = 'Canada'] \land \\
\exists p, q \ [ p \in \text{Person} \land p[\text{child}] = t[\text{nephew}] \land \\
q \in \text{Person} \land q[\text{child}] = p[\text{parent}] \land \\
\exists w \ [ w \in \text{Person} \land w[\text{parent}] = q[\text{parent}] \land w[\text{parent}] = q[\text{parent}] \land \\
w[\text{child}] \neq q[\text{child}] \land (t[\text{aunt}] = w[\text{child}] \lor \exists s \ [ s \in \text{Spouse} \land \\
s[\text{person1}] = w[\text{child}] \land s[\text{person2}] = t[\text{aunt}])}\}
\]
• Introduction

• Relational Calculus
  ▪ Syntax and Semantics
  ▪ Domain Independence and Safety
  ▪ Equivalence to RA

• Datalog
  ▪ Syntax and Semantics
  ▪ Recursion
  ▪ Negation
What is the Meaning of the Following?

\{ (x) \mid \neg \text{Person}(x, 'female', 'Canada') \} \\

\{ (x, y) \mid \exists z [\text{Spouse}(x, z) \land y = z] \} \\

\{ (x, y) \mid \exists z [\text{Spouse}(x, z) \land y \neq z] \} \\

- Person(id, gender, country)
- Parent(parent, child)
- Spouse(person1, person2)
Domain

• Let \( S \) be a schema, let \( I \) be an instance over \( S \), and let \( Q \) be an RC query over \( S \).
• The *active domain* (of \( I \) and \( Q \)) is the set of all the values that occur in either \( I \) or \( Q \).
• The query \( Q \) is evaluated over \( I \) with respect to a domain \( D \) that contains the active domain.
  – This is the set over which quantifiers range.
• We denote by \( Q^D(I) \) the result of evaluating \( Q \) over \( I \) relative to the domain \( D \).
Let \( S \) be a schema, and let \( Q \) be an RC query over \( S \).

We say that \( Q \) is *domain independent* if for every instance \( I \) over \( S \) and every two domains \( D \) and \( E \) that contain the active domain, we have:

\[
Q^D(I) = Q^E(I)
\]
Which One is Domain Independent?

\[
\{ (x) \mid \neg \text{Person}(x, \text{'female'}, \text{'Canada'}) \} \quad \text{Not DI}
\]

\[
\{ (x,y) \mid \exists z \ [\text{Spouse}(x,z) \land y=z] \} \quad \text{DI}
\]

\[
\{ (x,y) \mid \exists z \ [\text{Spouse}(x,z) \land y\neq z] \} \quad \text{Not DI}
\]

\[
\{ (x) \mid \exists z,w \ \text{Person}(x,z,w) \land \exists y \ [\neg \text{Likes}(x,y)] \} \quad \text{Not DI}
\]

\[
\{ (x) \mid \exists z,w \ \text{Person}(x,z,w) \land \forall y \ [\neg \text{Likes}(x,y)] \} \quad \text{DI}
\]

\[
\{ (x) \mid \exists z,w \ \text{Person}(x,z,w) \land \forall y \ [\neg \text{Likes}(x,y)] \land \exists y \ [\neg \text{Likes}(x,y)] \} \quad \text{DI}
\]

- Person(id, gender, country)
- Likes(person1, person2)
- Spouse(person1, person2)
Bad News...

• We would like be able to tell whether a given RA query is domain independent, – ... and then reject “bad queries”

• Alas, this problem is undecidable! – That is, there is no algorithm that takes as input an RC query and returns true iff the query is domain independent
Good News

Domain-independent RC has an **effective syntax**; that is:

– A syntactic restriction of RC in which every query is domain independent
  • Restricted queries are said to be **safe**
– Safety can be tested automatically (and efficiently)
– Most importantly, **for every domain independent RC query there exists an equivalent safe RC query!**
Safety

• We do not formally define the safe syntax in this course
• Details on the safe syntax can be found in the textbook *Foundations of Databases* by Abiteboul, Hull and Vianu
  – Example:
    • In $\exists x \varphi$, the variable $x$ should be guarded by $\varphi$
    • Every variable is guarded by $R(x_1,...,x_k)$
    • In $\varphi \land (x=y)$, the variable $x$ is guarded if and only if either $x$ or $y$ is guarded by $\varphi$
    • ... and so on
Outline

- Introduction
- Relational Calculus
  - Syntax and Semantics
  - Domain Independence and Safety
  - Equivalence to RA
- Datalog
  - Syntax and Semantics
  - Recursion
  - Negation
Example Revisited

\[ R(X,Y) \div S(Y) \]

**In RA:**
\[ \pi_X R \setminus \pi_X \left( (\pi_X R \times S) \setminus R \right) \]

**In RC:**
\[ \{ (X) \mid \exists Z \ [R(X,Z)] \land \forall Y \ [S(Y) \rightarrow R(X,Y)] \} \]
Another Example

- Person(id, gender, country)
- Parent(parent, child)
- Spouse(person1, person2)

\[
\{ (x) \mid \exists z, w \ Person(x, z, w) \land \forall y \ [\neg\text{Spouse}(x, y)] \}
\]

\[\pi_{id} \ Person \ \rho_{\text{person1}/id} \pi_{\text{person1}} \ Spouse\]
THEOREM: RA and domain-independent RC have the same expressive power.

More formally, on every schema $S$:

- For every RA expression $E$ there is a domain-independent RC query $Q$ such that $Q \equiv E$
- For every domain-independent RC query $Q$ there is an RA expression $E$ such that $Q \equiv E$
The proof has two directions:

1. Translate a given RA query into an equivalent RC query
2. Translate a given RC query into an equivalent RA query

Part 1 is fairly easy: induction on the size of the RA expression

Part 2 is more involved
Outline

• Introduction

• Relational Calculus
  ▪ Syntax and Semantics
  ▪ Domain Independence and Safety
  ▪ Equivalence to RA

• Datalog
  ▪ Syntax and Semantics
  ▪ Recursion
  ▪ Negation
Datalog

• Database query language
• “Clean” restriction of Prolog w/ DB access
  – Expressive & declarative:
    • *Set-of-rules* semantics
    • Independence of execution order
    • Invariance under logical equivalence
• Mostly academic implementations; some commercial instantiations
• Introduction
• Relational Calculus
  ▪ Syntax and Semantics
  ▪ Domain Independence and Safety
  ▪ Equivalence to RA
• Datalog
  ▪ Syntax and Semantics
  ▪ Recursion
  ▪ Negation
Example

- Person(id, gender, country)
- Parent(parent, child)
- Spouse(person1, person2)

Local(x) ← Person(x,y,'IL')
Relative(x,x) ← Person(x,y,z)
Relative(x,y) ← Relative(x,z),Parent(z,y)
Relative(x,y) ← Relative(x,z),Parent(y,z)
Relative(x,y) ← Relative(x,z),Spouse(z,y)
Invited(y) ← Relative('myself',y),Local(y)
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
Example

- Person(id, gender, country)
- Parent(parent, child)
- Spouse(person1, person2)

\[
\begin{align*}
\text{Local}(x) & \leftarrow \text{Person}(x,y, 'IL') \\
\text{Relative}(x,x) & \leftarrow \text{Person}(x,y,z) \\
\text{Relative}(x,y) & \leftarrow \text{Relative}(x,z), \text{Parent}(z,y) \\
\text{Relative}(x,y) & \leftarrow \text{Relative}(x,z), \text{Parent}(y,z) \\
\text{Relative}(x,y) & \leftarrow \text{Relative}(x,z), \text{Spouse}(z,y) \\
\text{Invited}(y) & \leftarrow \text{Relative}( 'myself' , y), \text{Local}(y)
\end{align*}
\]

EDB

IDB
Datalog Program

• 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

• A Datalog program is a finite set of Datalog rules
Logical Interpretation of a Rule

• Consider a Datalog rule of the form

\[ R(x) \leftarrow \psi_1(x,y), \ldots, \psi_m(x,y) \]

– Here, \( x \) and \( y \) are disjoint sequences of variables, and each \( \psi_i(x,y) \) is an atomic formula with variables among \( x \) and \( y \)

– Example: \( \text{TwoPath}(x_1, x_2) \leftarrow \text{Edge}(x_1, y), \text{Edge}(y, x_2) \)

• The rule stands for the logical formula

\[ R(x) \leftarrow \exists y \left[ \psi_1(x,y) \land \cdots \land \psi_m(x,y) \right] \]

– Example: if there exists \( y \) where \( \text{Edge}(x_1, y) \) and \( \text{Edge}(y, x_2) \) hold, then \( \text{TwoPath}(x_1, x_2) \) should hold
Syntactic Constraints

• We require the following from the rule

\[ R(x) \leftarrow \psi_1(x,y), \ldots, \psi_m(x,y) \]

1. Safety: every variable in \( x \) should occur in the body at least once

2. The predicate \( R \) must be an IDB predicate
   • (The body can include both EDBs and IDBs)

• Example of forbidden rules:
  – \( R(x,z) \leftarrow S(x,y), R(y,x) \)
  – \( \text{Edge}(x, y) \leftarrow \text{Edge}(x, z), \text{Edge}(x, y) \)
    • Assuming \( \text{Edge} \) is EDB
Semantics of Datalog Programs

- Let $S$ be a schema, $I$ an instance over $S$, and $P$ be a Datalog program over $S$
  - That is, all EDBs predicates belong to $S$
- The result of evaluating $P$ over $I$ is an instance $J$ over the IDB schema of $P$
- We give two definitions:
A chase procedure is a program of the following form:

\[
\text{Chase}(P,I)
\]

- \( J := \emptyset \)
- while(true) {
  - if(I \cup J satisfies all the rules of P) return J
  - Find a rule head(x) ← body(x,y) and tuples a, b such that I \cup J contains body(a,b) but not head(a)
  - \( J := J \cup \{\text{head}(a)\} \)
}
Nondeterminism

- Note: the chase is *underspecified*  
  - (i.e., not fully defined)

- There can be many ways of choosing the next violation to handle
  - And each choice can lead to new violations, and so on
  - We can view the choice of a new violation as *nondeterministic*
Example

- Person(id, gender, country)
- Parent(parent, child)
- Spouse(person1, person2)

Relative(x,x) $\iff$ Person(x,y,z)
Relative(x,y) $\iff$ Relative(x,z), Parent(z,y)
Invited(y) $\iff$ Relative('1',y)

<table>
<thead>
<tr>
<th>Person</th>
<th>Parent</th>
<th>Relative</th>
<th>Invited</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>F</td>
<td>IL</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>M</td>
<td>AU</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>F</td>
<td>NL</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>M</td>
<td>IL</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>F</td>
<td>FR</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>3</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Example

- Person(id, gender, country)
- Parent(parent, child)
- Spouse(person1, person2)

Relative(x,x) ⇔ Person(x,y,z)
Relative(x,y) ⇔ Relative(x,z),Parent(z,y)
Invited(y) ⇔ Relative('1',y)
Example

- Person(id, gender, country)
- Parent(parent, child)
- Spouse(person1, person2)

Relative(x,x) \iff Person(x,y,z)
Relative(x,y) \iff Relative(x,z), Parent(z,y)
Invited(y) \iff Relative('1',y)
Example

- Person(id, gender, country)
- Parent(parent, child)
- Spouse(person1, person2)

Relative(x,x) ⇐ Person(x,y,z)
Relative(x,y) ⇐ Relative(x,z), Parent(z,y)
Invited(y) ⇐ Relative('1', y)

<table>
<thead>
<tr>
<th>Person</th>
<th>Parent</th>
<th>Relative</th>
<th>Invited</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>F</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>M</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>F</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>M</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>F</td>
<td>5</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Person</th>
<th>Parent</th>
<th>Relative</th>
<th>Invited</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>F</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>M</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>F</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>M</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>F</td>
<td>5</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Person</th>
<th>Parent</th>
<th>Relative</th>
<th>Invited</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>F</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>M</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>F</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>M</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>F</td>
<td>5</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Person</th>
<th>Parent</th>
<th>Relative</th>
<th>Invited</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>F</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>M</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>F</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>M</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>F</td>
<td>5</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Person</th>
<th>Parent</th>
<th>Relative</th>
<th>Invited</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>F</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>M</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>F</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>M</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>F</td>
<td>5</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Person</th>
<th>Parent</th>
<th>Relative</th>
<th>Invited</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>F</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>M</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>F</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>M</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>F</td>
<td>5</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Person</th>
<th>Parent</th>
<th>Relative</th>
<th>Invited</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>F</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>M</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>F</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>M</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>F</td>
<td>5</td>
<td></td>
</tr>
</tbody>
</table>
Example

Relative(x,x) ⇐ Person(x,y,z)
Relative(x,y) ⇐ Relative(x,z), Parent(z,y)
Invited(y) ⇐ Relative('1', y)

- Person(id, gender, country)
- Parent(parent, child)
- Spouse(person1, person2)
Example

- Person(id, gender, country)
- Parent(parent, child)
- Spouse(person1, person2)

Relative(x,x) ⇐ Person(x,y,z)
Relative(x,y) ⇐ Relative(x,z), Parent(z,y)
Invited(y) ⇐ Relative('1',y)

<table>
<thead>
<tr>
<th>Person</th>
<th>Parent</th>
<th>Relative</th>
<th>Invited</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>F</td>
<td>IL</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>M</td>
<td>AU</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>F</td>
<td>NL</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>M</td>
<td>IL</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>F</td>
<td>FR</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td></td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td></td>
</tr>
</tbody>
</table>
Example

- Person(id, gender, country)
- Parent(parent, child)
- Spouse(person1, person2)

Relative(x,x) ← Person(x,y,z)
Relative(x,y) ← Relative(x,z), Parent(z,y)
Invited(y) ← Relative('1', y)

<table>
<thead>
<tr>
<th>Person</th>
<th>Parent</th>
<th>Relative</th>
<th>Invited</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>5</td>
<td>4</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
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
THEOREM: Let $S$ be a schema. For every Datalog program $P$ and instance $I$ over $S$:

there is a unique minimal model, and every chase returns that model

Proof: home assignment

This minimal model is the result, and it is denoted as $P(I)$
Outline

• Introduction

• Relational Calculus
  ▪ Syntax and Semantics
  ▪ Domain Independence and Safety
  ▪ Equivalence to RA

• Datalog
  ▪ Syntax and Semantics
  ▪ Recursion
  ▪ Negation
"Recursive" Program?

Local(x) ← Person(x,y,'IL')
Relative(x,x) ← Person(x,y,z)
Relative(x,y) ← Relative(x,z), Parent(z,y)
Relative(x,y) ← Relative(x,z), Parent(y,z)
Relative(x,y) ← Relative(x,z), Spouse(z,y)
Invited(y) ← Relative('myself',y), Local(y)

Local(x) ← Person(x,y,'IL')
Relative(x,x) ← Person(x,y,z)
Invited(y) ← Relative('myself',y), Local(y)

MayLike(x,y) ← Close(x,z), Likes(z,y)
Visit(x,y) ← MayLike(x,y)
Close(x,z) ← Visit(x,y), Visit(z,y)
Dependency Graph

- 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
Recursive?

Local(x) ← Person(x, y, 'IL')
Relative(x, x) ← Person(x, y, z)
Relative(x, y) ← Relative(x, z), Parent(z, y)
Relative(x, y) ← Relative(x, z), Parent(y, z)
Relative(x, y) ← Relative(x, z), Spouse(z, y)
Invited(y) ← Relative('myself', y), Local(y)
Local(x) ⇐ Person(x, y, 'IL')
Relative(x, x) ⇐ Person(x, y, z)
Invited(y) ⇐ Relative('myself', y), Local(y)
And This One?

\[
\begin{align*}
\text{MayLike}(x,y) & \leftarrow \text{Close}(x,z), \text{Likes}(z,y) \\
\text{Visit}(x,y) & \leftarrow \text{MayLike}(x,y) \\
\text{Close}(x,z) & \leftarrow \text{Visit}(x,y), \text{Visit}(z,y)
\end{align*}
\]
THEOREM: Datalog can express queries that RA (RC) cannot
(example: transitive closure of a graph)

*Proof not covered in the course*
THEOREM: Acyclic Datalog* has the same expressive power as the algebra \( \{\sigma_\leq, \pi, \times, \rho, U\} \)
(where \( \sigma_\leq \) means selection with a single equality)

* Without constants in rule heads

Proof: exercise

This fragment is often called “positive RA” or USPJ
(union-select-project-join)
Monotonicity

• Can Datalog express difference?
• Answer: No!
• Proof: Datalog is monotone
  – That is, if $I$ and $I'$ are such that every relation of $I'$ contains the corresponding relation of $I$, then $P(I) \subseteq P(I')$
Outline

• Introduction

• Relational Calculus
  ▪ Syntax and Semantics
  ▪ Domain Independence and Safety
  ▪ Equivalence to RA

• Datalog
  ▪ Syntax and Semantics
  ▪ Recursion
  ▪ Negation
What is the Semantics?

Adding negation to Datalog is not at all straightforward!

\[
\text{Buddy}(x,y) \leftarrow \text{Likes}(x,y), \neg \text{Parent}(y,x)
\]

\[
\text{Buddy}(x,y) \leftarrow \neg \text{Anti}(x,y), \text{Likes}(x,y)
\]
\[
\text{Anti}(x,y) \leftarrow \neg \text{Buddy}(x,y), \text{Suspects}(x,y)
\]

Likes('Avi','Alma')
Suspects('Avi','Alma')
Buddy('Avi','Alma')
Anti('Avi','Alma')
Negation in Datalog

• Various semantics have been proposed for supporting negation in Datalog
  – In a way that makes sense

• We will look at two:
  – *Semipositive* programs (restricted)
  – *Stratified* programs (standard)
Semipositive Programs

A *semipositive* program is a program where only EDBs may be negated

– Safety: every variable occurs in a positive literal

– Semantics: same as ordinary Datalog programs

Buddy(x, y) ← Likes(x, y), ¬Parent(y, x)
• 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 $E_0, \ldots, E_i$
  – For $i=1,\ldots,k$, every rule with head in $E_i$ can have negated body predicates only from $E_0, \ldots, E_{i-1}$
Example

- Person(id, gender, country)
- Fake(id)
- Parent(parent, child)
- Spouse(person1, person2)
- Likes(person1, person2)

\[
\begin{align*}
\text{RealPerson}(x) & \leftarrow \text{Person}(x,y,z), \neg \text{Fake}(x) \\
\text{Relative}(x,x) & \leftarrow \text{RealPerson}(x) \\
\text{Relative}(x,y) & \leftarrow \text{Relative}(x,z), \text{Parent}(z,y) \\
\text{Relative}(x,y) & \leftarrow \text{Relative}(x,z), \text{Parent}(y,z) \\
\text{Relative}(x,y) & \leftarrow \text{Relative}(x,z), \text{Spouse}(z,y) \\
\text{Buddy}(x,y) & \leftarrow \neg \text{Relative}(x,y), \text{Likes}(x,y) \\
\text{Buddy}(x,y) & \leftarrow \neg \text{Relative}(x,y), \text{Buddy}(x,z), \text{Buddy}(z,y)
\end{align*}
\]
Another Stratification?

- Person(id, gender, country)
- Fake(id)
- Parent(parent, child)
- Spouse(person1, person2)
- Likes(person1, person2)

\[
\begin{align*}
\text{RealPerson}(x) & \iff \text{Person}(x,y,z), \neg \text{Fake}(x) \\
\text{Relative}(x,x) & \iff \text{RealPerson}(x) \\
\text{Relative}(x,y) & \iff \text{Relative}(x,z), \text{Parent}(z,y) \\
\text{Relative}(x,y) & \iff \text{Relative}(x,z), \text{Parent}(y,z) \\
\text{Relative}(x,y) & \iff \text{Relative}(x,z), \text{Spouse}(z,y) \\
\text{Buddy}(x,y) & \iff \neg \text{Relative}(x,y), \text{Likes}(x,y) \\
\text{Buddy}(x,y) & \iff \neg \text{Relative}(x,y), \text{Buddy}(x,z), \text{Buddy}(z,y)
\end{align*}
\]
Semantics of Stratified Programs

• For i=1,...,k:
  – Compute the IDBs of the stratum $E_i$
  – Add computed IDBs to the EDBs

• Then, due to the definition of stratification, each $E_i$ can be viewed as semipositive

• *Does the result depend on the specific stratification of choice?*
  – Answer in the next slide
Theorems on Stratification (1)

• **THEOREM 1:** All stratifications are equivalent
  – That is, they give the same result on every input

• **THEOREM 2:** A program has a stratification if and only if its dependency graph does not contain a cycle with a “negated edge”
  – Dependency graph is defined as previously, except that edges can be labeled with negation
  – Hence, we can test for stratifiability efficiently, via graph reachability
• **Theorem 3**: Nonrecursive Datalog programs with negation are stratifiable
  – Via the topological order

• **Theorem 4**: Nonrecursive Datalog with negation has the same expressive power as the algebra \{\sigma, \pi, \times, \rho, \cup, \setminus\}
  – Extendable to RA if we add the predicates >, <
  – Again, we assume no constants in rule heads