Plan of the lecture

- More Datalog:
  - Safe queries
  - Datalog and relational algebra
- Recursive Datalog rules
- Semantics of recursive Datalog rules
- Problems with negation
- Stratified Datalog

Datalog syntax: rules

- A Datalog rule is an expression of the form
  \[ R_1 \leftarrow R_2 \text{ AND } \ldots \text{ AND } R_n \]
  where \( n \geq 1 \), \( R_1 \) is a relational atom, and \( R_2, \ldots, R_n \) are relational or arithmetic atoms, possibly preceded by NOT.
- \( R_1 \) is called the head of the rule and \( R_2, \ldots, R_n \) the body of the rule.
- \( R_2, \ldots, R_n \) are called subgoals.

Example

- Suppose we have a relation Person over schema (Name, Age, Address, Telephone). Then the following Datalog rule will define a relation which contains names of people aged over 18:
  \[ \text{Adult}(x) \leftarrow \text{Person}(x,y,z,u) \text{ AND } y \geq 18 \]

Datalog query

- A Datalog query is a finite set of Datalog rules
- If there is only one relation which appears as a head of a rule in the query, the tuples in that relation are taken as the answer to the query.
- For example,
  \[ \text{Parent}(x,y) \leftarrow \text{Mother}(x,y) \]
  \[ \text{Parent}(x,y) \leftarrow \text{Father}(x,y) \]
  defines Parent relation (using relations Father and Mother)
- If there is more than one relation appearing as a head, one of them is the main predicate to be defined and others are auxiliary.

Meaning of Datalog rules

- First approximation (non-recursive queries):
  - take the values of variables which make the body of the rule true (make each subgoal true; NOT R is true if R is false)
  - see what values the variables of the head take;
  - add the resulting tuple to the predicate in the head of the rule.
Example with negation

- Suppose we have a relation Person over schema (Name, Age, Address, Telephone).
  \[
  \text{Child}(x) \leftarrow \text{Person}(x,y,z,u) \text{ AND NOT}(y \geq 18)
  \]
- We take all \(<\text{name}, \text{age}, \text{addr}, \text{tel}>\) in Person for which it is also true that NOT(\(\text{age} \geq 18\)), and add \(<\text{name}>\) to Child.
- NOT(\(\text{age} \geq 18\)) is true if \(\text{age} \geq 18\) is false, so we add all tuples where \(\text{age} < 18\).

Safe queries

- We want the result of a query to be a finite relation.
- To ensure this, the following safety condition is required: *every variable that appears anywhere in the rule must appear in some non-negated relational subgoal.*
- The reason for this is that infinitely many values may satisfy an arithmetical subgoal (e.g. \(x > 0\)) and infinitely many values are NOT in some finite table of a relation \(R\).

Questions

- Which of the following rules have safety violations:
  - \(P(x,y) \leftarrow Q(x,y) \text{ AND NOT } R(x,y)\)
  - \(P(x,y) \leftarrow \text{NOT } Q(x,y) \text{ AND } y = 10\)
  - \(P(x,y) \leftarrow Q(x,z) \text{ AND NOT } R(w,x,z) \text{ AND } x < y\)
  - \(P(x,y) \leftarrow Q(x,z) \text{ AND } R(z,y) \text{ AND NOT } Q(x,y)\)

Questions

- Which tuples are in \(P\)?
  \(P(x,y) \leftarrow Q(x,z) \text{ AND } R(z,y) \text{ AND NOT } Q(x,y)\)
given that:
  - \(Q\) contains tuples \(<a,b>, <a,c>\)
  - \(R\) contains tuples \(<b,c>, <c,a>\)

Datalog and relational algebra

- Every relation definable in relational algebra is definable in Datalog.
- Again we assume that we have a relational name (predicate symbol) \(R\) for every basic relation \(R\).
- Then for every operation of relational algebra, we show how to write a corresponding Datalog query.

Union

- Union of \(R\) and \(S\):
  \[
  U(x_1, \ldots, x_n) \leftarrow R(x_1, \ldots, x_n) \\
  U(x_1, \ldots, x_n) \leftarrow S(x_1, \ldots, x_n)
  \]
Difference

- Difference of \( R \) and \( S \):
  \[
  D(x_1, \ldots, x_n) \leftarrow R(x_1, \ldots, x_n) \text{ AND NOT } S(x_1, \ldots, x_n)
  \]

Product

- Product of \( R \) and \( S \):
  \[
  P(x_1, \ldots, x_n, y_{n+1}, \ldots, y_k) \leftarrow R(x_1, \ldots, x_n) \text{ AND } S(y_{n+1}, \ldots, y_k)
  \]

Projection

- Suppose we want to project \( R \) on attributes \( x_1, \ldots, x_p \):
  \[
  P(x_1, \ldots, x_p) \leftarrow R(x_1, \ldots, x_p, y_{n+1}, \ldots, y_k)
  \]
  or
  \[
  P(x_1, \ldots, x_p) \leftarrow R(x_1, \ldots, x_p, _, \ldots, _n)
  \]

Selection

- Simple case: all conditions in the selection are connected by AND, for example \( \sigma\text{Age} > 18 \text{ AND Address} = "London"(\text{Person}) \)
  \[
  \text{Answer}(x,y,z,u) \leftarrow \text{Person}(x,y,z,u) \text{ AND } y > 18 \text{ AND } z = "London"
  \]
- If conditions are connected with OR, need more than one rule. For example, \( \sigma\text{Age} > 18 \text{ OR Address} = "London"(\text{Person}) \)
  \[
  \text{Answer}(x,y,z,u) \leftarrow \text{Person}(x,y,z,u) \text{ AND } y > 18
  \]
  \[
  \text{Answer}(x,y,z,u) \leftarrow \text{Person}(x,y,z,u) \text{ AND } z = "London"
  \]

Compound queries

- To translate an arbitrary algebraic expression, create a new predicate for every node in the query tree.
  - For example, to do \( \sigma \text{Name1} = \text{Name2} (R \times P) \):
    - Define predicate \( S = R \times P \)
    - Define \( \sigma \text{Name1} = \text{Name2} (S) \)

Recursion: motivating example

- Consider a database for London underground.
  - It describes lines, stations, station closures etc. (there may be stations closed on weekends, or because of technical problems or strikes).
  - Typical queries include:
    - is it possible to go from King’s Cross to Embankment?
    - which lines can be reached form King’s Cross?
Motivating example

- We can either compute and store this information for every station (recompute it every day because of station closures).
- Or, we can store the basic data (Links relation below) and compute answers to queries as they are asked.

<table>
<thead>
<tr>
<th>Line</th>
<th>Station</th>
<th>Next Station</th>
</tr>
</thead>
<tbody>
<tr>
<td>Central</td>
<td>Marble Arch</td>
<td>Bond St</td>
</tr>
<tr>
<td>Jubilee</td>
<td>Bond St</td>
<td>Green Park</td>
</tr>
<tr>
<td>Victoria</td>
<td>Green Park</td>
<td>Victoria</td>
</tr>
<tr>
<td>Victoria</td>
<td>Victoria</td>
<td>Pimlico</td>
</tr>
</tbody>
</table>

Motivating example

- However, in a relational database, given a relation Links, we cannot express a query “Is Pimlico reachable from Marble Arch?”.

<table>
<thead>
<tr>
<th>Line</th>
<th>Station</th>
<th>Next Station</th>
</tr>
</thead>
<tbody>
<tr>
<td>Central</td>
<td>Marble Arch</td>
<td>Bond St</td>
</tr>
<tr>
<td>Jubilee</td>
<td>Bond St</td>
<td>Green Park</td>
</tr>
<tr>
<td>Victoria</td>
<td>Green Park</td>
<td>Victoria</td>
</tr>
<tr>
<td>Victoria</td>
<td>Victoria</td>
<td>Pimlico</td>
</tr>
</tbody>
</table>

Recursive queries

- Reachability in a graph is a typical recursive property.
- It cannot be expressed in relational calculus or relational algebra given an Edge relation for the graph.
- We can write a query which expresses “reachable in one step”, “reachable in two steps”, and so on, but not simply “reachable”.
- Another example: given a Parent relation, write a query which finds ancestors of a given person.
- Again, in relational algebra or calculus we can find parents, grandparents and so on, but not all ancestors.

Example recursive program

Reachable (x,x) ←  
Reachable (x,y) ← Links(u,z,y) AND Reachable (x,z)  
• We use the database relation Links to define relation Reachable, which is not stored in the database.
• To compute the set of stations reachable from King’s Cross, we add to this program  
Answer(y) ← Reachable(“King’s Cross”, y)

Extensional and intensional predicates

- To distinguish relations which are in the database and relations which are being defined by Datalog rules:
  - Extensional predicates: predicates whose relations are stored in a database
  - Intensional predicates: defined by Datalog rules
- EDB – extensional database – collection of extensional relations
- IDB – intensional database – collection of intensional relations

Three ways to give semantics of recursive Datalog programs

- Minimal relations (minimal models)
- Provability semantics
- Fixpoint semantics
For the time being, assume that we do not have negation on IDB predicates
Minimal relations

- Datalog programs are logical descriptions of new relations. The answer to the Datalog query is the smallest relation which satisfies all the stated properties.
- Each rule
  \[ R(x) \leftarrow R_1(x) \land \ldots \land R_n(x) \]
  corresponds to a logical property
  \[ \forall x_1 \ldots \forall x_m (R_1(x) \land \ldots \land R_n(x) \rightarrow R(x)) \]
  where \( x_1, \ldots, x_m \) are all the variables occurring in the rule and \( x \) some subsequence of \( x_1, \ldots, x_m \).

Example

- A program
  \[ \text{Ancestor}(x,y) \leftarrow \text{Parent}(x,y) \]
  \[ \text{Ancestor}(x,y) \leftarrow \text{Parent}(x,z) \land \text{Ancestor}(z,y) \]
- corresponds to logical properties
  \[ P_1 \forall x \forall y (\text{Parent}(x,y) \rightarrow \text{Ancestor}(x,y)) \]
  \[ P_2 \forall x \forall y \forall z (\text{Parent}(x,z) \land \text{Ancestor}(z,y) \rightarrow \text{Ancestor}(x,y)) \]

Example

- Suppose \( \text{Parent} \) contains just two pairs:
  \( \text{Parent}(\text{Anne}, \text{Bob}), \text{Parent}(\text{Bob}, \text{Chris}) \)
- Because of \( P_1 \), \( \text{Ancestor} \) should contain the same pairs:
  \( \text{Ancestor}(\text{Anne}, \text{Bob}), \text{Ancestor}(\text{Bob}, \text{Chris}) \)
- Because of \( P_2 \), we also need to add \( \text{Ancestor}(\text{Anne}, \text{Chris}) \)
  to satisfy
  \[ \forall x \forall y \forall z (\text{Parent}(x,z) \land \text{Ancestor}(z,y) \rightarrow \text{Ancestor}(x,y)) \]
  \( \text{Parent}(\text{Anne}, \text{Bob}) \land \text{Ancestor}(\text{Bob}, \text{Chris}) \rightarrow \text{Ancestor}(\text{Anne}, \text{Chris}) \)

Programs as proofs

- Proof-theoretic way of looking at Datalog programs:
  - for which tuples can we logically prove that they are in Ancestor relation (using Parent relation and the program rules).
  - Happens to be the same tuples as in the minimal Ancestor relation.

Fixpoint semantics of programs

- Start assuming that all IDB predicates are empty.
- Construct larger and larger IDB relations by:
  - Fire rules to add a tuples to IDB relations
  - Use tuples added to IDB relations in the previous round to add a new tuples to IDB relations
- Continue firing rules until no new tuples are added (reached a fixpoint). If rules are safe, there will be finitely many tuples which satisfy the body of the rule, so fixpoint will be reached after finitely many rounds.
- This happens to give the same answer as "what is the minimal relation satisfying the properties" and "for which tuples can we prove that they are in Ancestor relation".

Example: fixpoint construction

\[ \text{Ancestor}(x,y) \leftarrow \text{Parent}(x,y) \]
\[ \text{Ancestor}(x,y) \leftarrow \text{Parent}(x,z), \text{Ancestor}(z,y) \]
- Start: Ancestor = \{ \}, Parent = \{<a,b>,<b,c>,<c,d>\}
- 1st round: Ancestor = \{<a,b>,<b,c>,<c,d>,<a,c>,<b,d>\}
- 2nd round: Ancestor = \{<a,b>,<b,c>,<c,d>,<a,c>,<b,d>,<a,d>\}
- 3rd round: Ancestor = \{<a,b>,<b,c>,<c,d>,<a,c>,<b,d>,<a,d>\}
- 4th round: no new tuples in Ancestor.
Negation

- Problem with negation: may not be a unique minimal solution; no clear semantics.
- Example: EDB = \{R\} and IDB = \{P,Q\}

\[
\begin{align*}
P(x) &\leftarrow R(x) \text{ AND NOT } Q(x) \\
Q(x) &\leftarrow R(x) \text{ AND NOT } P(x)
\end{align*}
\]

Suppose R = \{<a>\}. Then either
- P = \{<a>\} and Q empty, or
- Q = \{<a>\} and P empty.

No unique solution. Can’t say if P(a) holds or Q(a) holds.

Stratified Datalog with negation

- The idea is to break cycles as in the example before, when to evaluate IDB predicate P we need to know what is the negation of IDB predicate Q, and vice versa (P is defined using NOT Q and Q is defined using NOT P).
- Solution: outlaw cycles in dependencies on negative IDB predicates.

What does “depend” mean

- If R is the head of a rule where P is in the body, R depends on P.
- If R is the head of a rule where P is in the body, and P depends on S, then R depends on S (transitive relation).
- We draw a dependency graph for IDB predicates.

Strata

- In a stratified program, IDB predicates are divided into strata.
- Stratum of a predicate is the maximal number of negative arcs on a dependency path starting at that predicate.
Example

• The program below is stratified.
• Stratum 0 = \{S, V\}
• Stratum 1 = \{P, Q\}
• Stratum 2 = \{R\}

\[
\begin{align*}
R(x) & \leftarrow P(x) \land \neg Q(x) \\
Q(x) & \leftarrow \neg V(x) \land E(x) \\
P(x) & \leftarrow \neg S(x) \land E(x)
\end{align*}
\]

In other words

• stratum 0: do not depend on any negated IDB predicates
• stratum 1: depend on negated IDB predicates from stratum 0;
• stratum 2: depend on negated IDB predicates from stratum 1,
• …
• stratum n: depend on IDB predicates from stratum n-1.

Evaluating stratified Datalog programs

• Stratified Datalog programs have the following operational semantics:
  – First compute all IDB predicates in stratum 0 (using the usual fixpoint strategy)
  – …
  – Using IDB predicates from stratum n, compute IDB predicates from stratum n+1.
• This produces unique minimal solutions for all IDB predicates.

Informal coursework

• Is the following program stratified (EDB = \{S\}):
  \[
  \begin{align*}
  Q(x) & \leftarrow \neg P(x) \land R(x) \\
P(x) & \leftarrow \neg R(x) \land S(x) \\
R(x) & \leftarrow S(x)
  \end{align*}
  \]
• Is the following program stratified (EDB = \{S\}):  
  \[
  \begin{align*}
  R(x) & \leftarrow Q(x) \\
Q(x) & \leftarrow R(x) \\
R(x) & \leftarrow S(x) \land \neg Q(x)
  \end{align*}
  \]
• For the stratified program, compute P, Q and R given that S contains \{<a>,<b>\}.

Informal coursework

• A database of fictitious company contains three relations:
  – GOODS over schema \{Producer, ProductCode, Description\}
  – DELIVERY over schema \{Producer, ProductCode, Branch#, Stock#\}
  – STOCK over schema \{Branch#, Stock#, Size, Colour, SellPrice, CostPrice, DateIn, DateOut\}.

Define in Datalog

• Query 1: find all producers who supply goods.
• Query 2: find all producers who have delivered goods to any branch of the company.
• Query 3: find all goods delivered to branch L1 still in stock (here, L1 is a value in the attribute domain of Branch#, and products in stock have value InStock for the DateOut attribute).
• Query 4: find Producer, ProductCode, Description for all goods sold at the same day they arrived at any branch.
• Query 5: find Branch#, Size, Colour, SellPrice for all dresses which have not yet been sold (dress is a value in the attribute domain of Description).
Reading

- Ullman, Widom, chapter 10
- Abiteboul, Hull, Vianu chapter 12.