Database Management Systems
Course 236363

Tutorial 13:
Summary

Datalog with negation

Monotonicity

- Datalog (without negation) is monotonic
  - That is, if we evaluate a program $P$ on two databases $D \subseteq D'$ then $P(D) \subseteq P(D')$
- Enabling negation in Datalog increases expressiveness as it enables us to phrase queries that are not monotonic.
- However, a new semantics should be defined.

Example

- Let us consider the schema consists of two relation symbols:
  - $\text{Node}(x)$
  - $\text{Edge}(x,y)$
- A database over this schema represents a directed graph (in the natural way).

Example – cont’d

- Define the query that outputs the binary relation $\text{NoEdge}$ such that $\text{NoEdge}(x,y)$ if there is no edge between nodes $x$ and $y$.
- Solution:
  - We certainly need negation (why?)
  - 1st try: $\text{NoEdge}(x,y) \leftarrow \neg \text{Edge}(x,y)$
  - Incorrect! What about safety?
  - Correct query:
    - $\text{NoEdge}(x,y) \leftarrow \neg \text{Edge}(x,y), \text{Node}(x), \text{Node}(y)$

Semipositive Vs. Stratified Negation

- Note that in the previous query we have negated an EDB.
- There are two different extensions of Datalog:
  - Semipositive Datalog and
  - stratified Datalog
- In semipositive Datalog only EDBs may be negated.
- The semantics is the same as Datalog
- As usual, we should make sure that our programs are safe.
Example

- Define the query that outputs the binary relation NoPath such that NoPath(x,y) if there is no path between nodes x and y.
- A path can be of length more than one
- Is semipositive sufficient?
  - Apparently not.
- We can find all paths by the rules:
  - Path(x,y) ← Edge(x,y)
  - Path(x,y) ← Edge(x,z), Path(z,y)

Example – cont’d

- If we assume that Path is given as an EDB then the following semipositive program does the desired:
  - NoPath(x,y) ← Node(x), Node(y), ¬Path(x,y)
- However, Path is not given as an EDB.
- The following program is not a semipositive program:
  - Path(x,y) ← Edge(x,y)
  - Path(x,y) ← Edge(x,z), Path(z,y)
  - NoPath(x,y) ← Node(x), Node(y), ¬Path(x,y)

Stratified Negation

- We define a stratification over the IDBs of the program.
- The semantics of stratified programs is defined as follows:
  - For i=1,...,k:
    - Compute the IDBs of the stratum Ei
    - Add computed IDBs to the EDBs
- In our example the IDB Path is computed before NoPath

Non-recursive Datalog Vs. RA

- Non-recursive (NR) datalog without negation is equivalent to \{σ, π, U, ×, ρ\}
  - Everything that can be done with \{σ, π, U, ×, ρ\}
    - can be done with NR-Datalog:
      - \σ\ by giving variables the same name
      - \pi\ by eliminating variables in the head of the rule
      - \u\ by writing more than one rule to define the same relation
      - \×\ by writing more than one atom in the RHS (each atom with different variables then the other).
      - \ρ\ by changing variable’s name
  - With negation:
    - \{σ, π, U, ×, ρ, \}\}

Relational Algebra

- Prove/refute: it is possible to express the operator ÷ using operators from the set \{π, ρ, σ, U, ×\}.
  - To prove this we need to write an explicit expression that is equivalent to ÷
  - To refute, we need to find a property that is preserved by \{π, ρ, σ, U, ×\} but is not preserved by ÷ (or vice versa).
  - We show that by induction on the structure of the RA expression
Relational Algebra – cont’d

• In this case, we refute the argument.
• We use the monotonicity property:
  – the operators (∩, ∪, σ, π) are monotonic,
  – while ÷ is not.
• Monotonicity:
  – A unary operator $\zeta$ is monotonic if:
    • If $R \subseteq R'$ then $\zeta(R) \subseteq \zeta(R')$.
  – A binary operator $\circ$ is monotonic if:
    • If $R_1 \subseteq R_1'$ and $R_2 \subseteq R_2'$ then $R_1 \circ R_2 \subseteq R_1' \circ R_2'$.
• Note that the same argument is valid when we replace ÷ with difference \.

Decompositions and Normal Forms

BCNF – cont’d

• Let us consider the schema $(R, F)$ where:
  – $R = \{Id, Name, Address\}$
  – $F = \{Id \rightarrow Name, Name \rightarrow Address\}$
  – Its only key is $\{Id\}$.
  – It is not in BCNF due to the second FD.
  – We can decompose it so that each of the resulting schemas will be in BCNF.
  – With respect to the set of FDs implied on each of the elements in the decomposition.
  – If we decompose it into $R_1(\{Id, Name\})$ and $R_2(\{Name, Address\})$ then
    – The set of FDs implied on $R_1$ is $F_1(=\{Id \rightarrow Name\})$ and on $R_2$ is $F_2(=\{Name \rightarrow Address\})$
    – Each $(R_i, F_i)$ is in BCNF:
      • In $(R_1, F_1)$ the key is Id and thus the FD in $F_1$ is ok
      • In $(R_2, F_2)$ the key is Name.

RDF

• Given the following RDF graph:

```
<table>
<thead>
<tr>
<th>dbr:C1</th>
<th>dbp:mayor</th>
<th>dbr:M1</th>
</tr>
</thead>
<tbody>
<tr>
<td>dbr:C22</td>
<td>dbp:birthPlace</td>
<td>dbr:C22</td>
</tr>
<tr>
<td>dbr:M22</td>
<td>dbp:birthPlace</td>
<td>dbr:C1</td>
</tr>
<tr>
<td>dbr:C33</td>
<td>rdf:type</td>
<td>dbr:person</td>
</tr>
<tr>
<td>dbr:M333</td>
<td>dbp:mayor</td>
<td>dbr:M333</td>
</tr>
<tr>
<td>dbr:M333</td>
<td>dbp:birthPlace</td>
<td>dbr:C333</td>
</tr>
</tbody>
</table>
```

• What will the following query return?

```sql
SELECT ?c, ?p WHERE {
    { ?c dbp:mayor ?m. }
    UNION {
    }
}
```

RDF Vs. RDFS
**Given the following RDF graph:**

- RDF

**What will the following query return?**

**RDF**

```
SELECT ?c, ?m WHERE {
  ?c dbp:mayor ?m.
  MINUS {
  }
}
```

```
?c  C1, ?m  M1
?c  C22, ?m  M22
?c  C333, ?m  M333
```

**RDFS**

```
SELECT ?c, ?m WHERE {
  ?c dbo:mayor ?m.
  OPTIONAL {
  }
  MINUS {
  }
}
```

```
?c  C1, ?m  M1
?c  C22, ?m  M22
?c  C333, ?m  M333
```