Name

CWID

Homework Assignment
1

February 16, 2016

CS520
Results

Please leave this empty!

1.1		1.2		1.3		1.4		1.5		1.6					
1.7		1.8		1.9		1.10		1.11		1.12		1.13		Sum	

Instructions

- Try to answer all the questions using what you have learned in class
- The assignment is not graded
- There is a theoretical and practical part
- **When writing a query, write the query in a way that it would work over all possible database instances and not just for the given example instance!**

Lab Part

- This part of the assignment helps you to practice the techniques we have introduced in class. For this assignment we are focusing on:
    - Loading a dataset into a database
    - Getting used to writing Datalog queries using the DLV system

Hospital Dataset

- We have uploaded a hospital dataset to the course webpage: `!http://cs.iit.edu/~cs520/hospital.csv`
- The database instance is stored in a CSV file
- The schema of this database contains a single table with attributes
    - **providernumber**
    - **hospitalname**
    - **address1**
    - **address2**
    - **address3**
    - **city**
    - **state**
    - **zip**
    - **country**
    - **phone**
    - **hospitaltype**
    - **hospitalowner**
    - **emergencyservice**
    - **condition**
    - **measurecode**
    - **measurename**
    - **score**
    - **sample**
    - **stateavg**

  The following constraints (functional dependencies) have been defined for the dataset:

$$
\begin{aligned}
e_0 &: zip \rightarrow city \\
e_1 &: zip \rightarrow state \\
e_2 &: phone \rightarrow zip \\
e_3 &: phone \rightarrow city \\
e_4 &: phone \rightarrow state \\
e_7 &: providernumber, measurecode \rightarrow stateavg \\
e_8 &: state, measurecode \rightarrow stateavg
\end{aligned}
$$

Part 1.1 Create Schema and Load Dataset (Total: 0 Points)

- Load the database into your favorite database / NoSQL store / distributed file system. Use a system you are comfortable with and where you would know how to write the queries required for the next questions (have a look at these questions first).
- As an example here are the steps outlined for Postgres
  - Run the DDL to create a (single table) schema for the dataset
  - Use the loader utility of your database (e.g., `COPY` command in Postgres) to load the content of the CSV file into your table
- In the next homework assignment we will perform various cleaning tasks using this dataset.

Part 1.2 Download DLV and get used to the system (Total: 0 Points)

- DLV is a logic programming system that supports Datalog (and more). Download DLV from `http://www.dlvsystem.com/dlv/`
- input to dlv is a `.dlv` file (text) which stores facts (the edb) and Datalog rules.
- DLV uses the following syntactical conventions:
  - Facts are written as `Q(c1, \ldots, cn).` where each c_i is a constant.
  - Rules are written as `Q(X) :- R1(X1), ... , Rn(Xn).` where X's can contain constants and variables.
  - Variable names start with an uppercase character (e.g., $X, Y, Name, \ldots$), constants with a lower case character $(x, y, chicago, \ldots)$.
  - If the value of a variable is not used by the query (the variable does not occur in the head and does only occur once in the body), you may replace the variable with an underscore. For example, you may write $Q(X) : -R(X, \_)$. instead of $Q(X) : -R(X, Y)$.
- Running dlv:
  - Open a terminal and run `dlv test.dlv` to run DLV over the input file `test.dlv`
  - DLV will show all edb atoms defined in this file and all idb atoms that can be computed from these edb atoms based on the rules in the file.
  - To only show certain predicates use the `-filter=predicate` option on the commandline
    - ∗ e.g., `dlv -filter=Q test.dlv` to show only the instance of predicate Q

Part 1.3 Create your first edb instance and run your first query (Total: 0 Points)

- Create a text file `hop.dlv` file
- Insert the following facts into the file:

```
hop(a,b).
hop(b,c).
hop(a,c).
hop(c,d).
```

Note that this is the example graph that was used in class.

- Run dlv to check that this step was done correctly: `dlv hop.dlv`

You should see output like this:

```
dhcp8:~ lord_pretzel: dlv hop.dlv

DLV [build BEN/Dec 17 2012   gcc 4.2.1 (Apple Inc. build 5666) (dot 3)]

{hop(a,b), hop(a,c), hop(b,c), hop(c,d)}
```

- Now add a Datalog rule to the file: $Q(X) : -hop(X, Y)$. This rule returns nodes that are the starting points of edges.

```
hop(a,b).
hop(b,c).
hop(a,c).
hop(c,d).

Q(X) :- hop(X,Y).
```

- Run dlv to check that this step was done correctly: `dlv hop.dlv`

```
dhcp8:~ lord_pretzel: dlv hop.dlv
DLV [build BEN/Dec 17 2012   gcc 4.2.1 (Apple Inc. build 5666) (dot 3)]

{hop(a,b), hop(a,c), hop(b,c), hop(c,d), Q(a), Q(b), Q(c)}
```

Note that DLV now also lists the atoms of predicate Q that can be derived based on the edb instance. To only show the query result (predicate Q) but not the edb instance use the filter predicate:

```
dhcp8:~ lord_pretzel: dlv -filter=Q hop.dlv
DLV [build BEN/Dec 17 2012   gcc 4.2.1 (Apple Inc. build 5666) (dot 3)]

{Q(a), Q(b), Q(c)}
```

Part 1.4 Run some additional hop queries (Total: 0 Points)

Write the following queries over the hop relation.

- Return all nodes in the graph
- Return all pairs of nodes that can be reached from each other through paths of length 2 (also do paths of length 3 and 4).
- Return all edges (x, y) for which a reversed edge (y, x) exists
- Return all edges (x, y) for which no reversed edge (y, x) exists

Part 1.5 Translate edb instance from the theory part (Total: 0 Points)

Create a file `transportation.dlv` and add the edb instance from the theory part.

Part 1.6 Run the queries from the theory part (Total: 0 Points)

Run the queries from the theory part using DLV and the `transportation.dlv` file you have created previously.

Theory Part

- This part of the assignment helps you to practice the techniques we have introduced in class.
- In this assignment we focus on Datalog queries and modelling of constraints using the logical notation introduced in class.

Consider the following transportation database schema and example instance:

road

fromCity	toCity	length
Chicago	Evanston	13
Chicago	Evanston	14
Chicago	Oak Park	8
Oak Park	Naperville	20
Chicago	Naperville	18

city

name	gasPrice	population
Chicago	1.80	5,000,000
Evanston	1.90	300,000
Oak Park	1.50	500,000
Naperville	1.60	22,000

train

fromCity	toCity	price
Chicago	Evanston	20
Chicago	Oak Park	34
Oak Park	Naperville	12

Hints:

- Attributes with black background form the primary key of a relation
- The attributes *fromCity* and *toCity* of relation *road* are both foreign keys to relation *city*
- The attributes *fromCity* and *toCity* or relation *trans* are both foreign keys to relation *city*

Part 1.7 Datalog (Total: 0 Points)

Question 1.7.1 Population of Chicago (0 Points)

Write a Datalog program that returns the population of Chicago.

Solution

$$Q(X) : -city(chicago, Y, X).$$

Question 1.7.2 Long roads (0 Points)

Write a Datalog program that returns the direct roads between cities that are at least 10 miles long.

Solution

$$Q(X, Y) : -road(X, Y, Z), Z > 9.$$

Question 1.7.3 Connected cities (0 Points)

Write a Datalog program that returns pairs of cities that can be reached from each other with a direct road or train connection.

Solution

$$Q(X, Y) : -road(X, Y, Z).$$
$$Q(X, Y) : -train(X, Y, Z).$$

Question 1.7.4 Large connected cities (0 Points)

Write a Datalog program that returns pairs of cities that can be reached from each other with a direct road where both cities have a population larger than 100,000 people.

Solution

$$LargeCity(X) : -city(X, \_, Y), Y > 100,000.$$
$$Q(X, Y) : -road(X, Y, \_), LargeCity(X), LargeCity(Y).$$

Question 1.7.5 Train and roads (0 Points)

Write a Datalog program that computes which cities are directly reachable from each via train **and** road.

Solution

$$Q(X, Y) : -road(X, Y, \_), train(X, Y, \_).$$

Question 1.7.6 Reachability of cities (0 Points)

Write a Datalog program that computes which cities are reachable from each other. To reach a city from another city one has to either take a train connecting these cities or a road. Note that it may require multiple steps to reach one city from another. Furthermore, for this question assume that roads and trains are running in both directions even if the database only contains only one direction. For example, in the example instance there is a train from *Oak Park* to *Chicago*.

Solution

$$\begin{aligned}
oneHop(X,Y) &: -road(X,Y,Z).\\
oneHop(X,Y) &: -road(Y,X,Z).\\
oneHop(X,Y) &: -train(X,Y,Z).\\
oneHop(X,Y) &: -train(Y,X,Z).\\
reach(X,Y) &: -oneHop(X,Y).\\
reach(X,Y) &: -reach(X,Z), oneHop(Z,Y).
\end{aligned}$$

Question 1.7.7 Train lines (0 Points)

Write a Datalog program that computes which cities are reachable from each other via train with at most 2 transfers.

Solution

$$\begin{aligned}
oneHop(X,Y) &: -train(X,Y,Z).\\
oneHop(X,Y) &: -train(Y,X,Z).\\
twoHops(X,Y) &: -oneHop(X,Z), oneHop(Z,Y).\\
threeHops(X,Y) &: -twoHop(X,Z), oneHop(Z,Y).\\
reach(X,Y) &: -oneHop(X,Y).\\
reach(X,Y) &: -twoHops(X,Y).\\
reach(X,Y) &: -threeHops(X,Y).
\end{aligned}$$

Question 1.7.8 Train lines (0 Points)

Translate the program from the previous question into relational algebra and SQL

Solution

```
WITH
  oneHop AS (SELECT fromCity, toCity FROM train
             UNION ALL
             SELECT toCity AS fromCity, fromCity AS toCity FROM train),

  twoHops AS (SELECT t.fromCity o.toCity
             FROM oneHop t JOIN oneHop o
             WHERE t.toCity=o.fromCity)

  threeHops AS (SELECT t.fromCity o.toCity
             FROM twoHop t JOIN oneHop o
             WHERE t.toCity=o.fromCity)

SELECT DISTINCT fromCity, toCity
FROM
  (SELECT * FROM oneHops
   UNION ALL
   SELECT * FROM twoHops
   UNION ALL
   SELECT * FROM threeHops) hops
```

$$\begin{aligned}
oneHop &\leftarrow train \cup \rho_{fromCity \leftarrow toCity, toCity \leftarrow fromCity}(train) \\
twoHops &\leftarrow \pi_{fromCity,toCity}(\rho_{joinCity \leftarrow toCity}(oneHop) \bowtie \rho_{joinCity \leftarrow fromCity}(oneHop)) \\
threeHops &\leftarrow \pi_{fromCity,toCity}(\rho_{joinCity \leftarrow toCity}(twoHop) \bowtie \rho_{joinCity \leftarrow fromCity}(oneHop)) \\
q &\leftarrow oneHop \cup twoHops \cup threeHops
\end{aligned}$$

Part 1.8 Constraints (Total: 0 Points)

Question 1.8.1 Translation into logical formalism (0 Points)

Translate the functional dependencies e_0 to e_8 from the lab part into the first-order logical representation that was introduced in class.

Solution

$$\begin{array}{ll}
e_0: & \forall \mathit{zip}, \mathit{city}1, \mathit{city}2 : \mathit{hospital}(\mathit{zip}, \mathit{city}1) \land \mathit{hospital}(\mathit{zip}, \mathit{city}2) \rightarrow \mathit{city}1 = \mathit{city}2 \\
e_1: & \forall \mathit{zip}, \mathit{state}1, \mathit{state}2 : \mathit{hospital}(\mathit{zip}, \mathit{state}1) \land \mathit{hospital}(\mathit{zip}, \mathit{state}2) \rightarrow \mathit{state}1 = \mathit{state}2 \\
e_2: & \forall \mathit{phone}, \mathit{zip}1, \mathit{zip}2 : \mathit{hospital}(\mathit{phone}, \mathit{zip}2) \land \mathit{hospital}(\mathit{phone}, \mathit{zip}2) \rightarrow \mathit{zip}1 = \mathit{zip}2 \\
e_3: & \forall \mathit{phone}, \mathit{city}1, \mathit{city}2 : \mathit{hospital}(\mathit{phone}, \mathit{city}2) \land \mathit{hospital}(\mathit{phone}, \mathit{city}2) \rightarrow \mathit{city}1 = \mathit{city}2 \\
e_4: & \forall \mathit{phone}, \mathit{state}1, \mathit{state}2 : \mathit{hospital}(\mathit{phone}, \mathit{state}2) \land \mathit{hospital}(\mathit{phone}, \mathit{state}2) \rightarrow \mathit{state}1 = \mathit{state}2 \\
e_6: & \forall \mathit{pnum}, \mathit{mcode}, \mathit{avg}1, \mathit{avg}2 : \mathit{hospital}(\mathit{pnum}, \mathit{mcode}, \mathit{avg}1) \land \mathit{hospital}(\mathit{pnum}, \mathit{mcode}, \mathit{avg}2) \rightarrow \mathit{avg}1 = \mathit{avg}2 \\
e_7: & \forall \mathit{state}, \mathit{mcode}, \mathit{avg}1, \mathit{avg}2 : \mathit{hospital}(\mathit{state}, \mathit{mcode}, \mathit{avg}1) \land \mathit{hospital}(\mathit{state}, \mathit{mcode}, \mathit{avg}2) \rightarrow \mathit{avg}1 = \mathit{avg}2
\end{array}$$

Question 1.8.2 Translation into logical formalism (0 Points)

Translate the primary and foreign key constraints of the transportation schema present before into the first-order logical representation that was introduced in class.

Solution

Note that the primary key constrain on relation road trivially holds under set semantics (all attributes).

$$\begin{array}{ll}
PK(\mathit{city}): & \forall \mathit{name}, gP1, gP2, \mathit{ppl}1, \mathit{ppl}2 : \mathit{city}(\mathit{name}, gP1, \mathit{ppl}1) \land \mathit{city}(\mathit{name}, gP2, \mathit{ppl}2) \rightarrow gP1 = gP2 \land \mathit{ppl}1 = \mathit{ppl}2 \\
PK(\mathit{train}): & \forall fCity, tCity, p1, p2 : \mathit{train}(fCity, tCity, p1) \land \mathit{train}(fCity, tCity, p2) \rightarrow p1 = p2 \\
FK_1(\mathit{road}): & \forall fCity, t, l : \mathit{road}(fCity, t, l) \rightarrow \exists gPrice, \mathit{ppl} : \mathit{city}(fCity, gPrice, \mathit{ppl}) \\
FK_2(\mathit{road}): & \forall f, tCity, l : \mathit{road}(f, tCity, l) \rightarrow \exists gPrice, \mathit{ppl} : \mathit{city}(tCity, gPrice, \mathit{ppl}) \\
FK_1(\mathit{train}): & \forall fCity, t, l : \mathit{train}(fCity, t, l) \rightarrow \exists gPrice, \mathit{ppl} : \mathit{city}(fCity, gPrice, \mathit{ppl}) \\
FK_2(\mathit{train}): & \forall f, tCity, l : \mathit{train}(f, tCity, l) \rightarrow \exists gPrice, \mathit{ppl} : \mathit{city}(tCity, gPrice, \mathit{ppl})
\end{array}$$

Question 1.8.3 Translation into denial constraints (0 Points)

Translate the functional dependencies e_0 to e_8 from the lab part into denial constraints.

Solution

$$
\begin{array}{ll}
e_0: & \forall\neg(hospital(zip, city1) \wedge hospital(zip, city2) \wedge city1 \neq city2) \\
e_1: & \forall\neg(hospital(zip, state1) \wedge hospital(zip, state2) \wedge state1 \neq state2) \\
e_2: & \forall\neg(hospital(phone, zip2) \wedge hospital(phone, zip2) \wedge zip1 \neq zip2) \\
e_3: & \forall\neg(hospital(phone, city2) \wedge hospital(phone, city2) \wedge city1 \neq city2) \\
e_4: & \forall\neg(hospital(phone, state2) \wedge hospital(phone, state2) \wedge state1 \neq state2) \\
e_6: & \forall\neg(hospital(pnum, mcode, avg1) \wedge hospital(pnum, mcode, avg2) \wedge avg1 \neq avg2) \\
e_7: & \forall\neg(hospital(state, mcode, avg1) \wedge hospital(state, mcode, avg2) \wedge avg1 \neq avg2)
\end{array}
$$

Question 1.8.4 Creating denial constraints (0 Points)

Create denial constraints over the transportation schema that encode the following restrictions (note: it may be necessary to use more than one constraint to express some of the restrictions):

1. The gas price of cities with over 200,000 inhabitats (population attribute) is always above or equals to 1.5
2. The difference in length between two roads connecting the same cities is never more than 10 miles
3. The direct train route between two cities is always more expensive than each individual train on a route with one intermediate stop. E.g., the train $(Chicago, Naperville)$ has to be more expensive than the trains $(Chicago, OakPark)$ and $(OakPark, Naperville)$

Solution

Restriction 1

$$\forall\neg(city(city, inhabitats, gasprice) \wedge inhabitats > 200,000 \wedge gasprice < 1.5)$$

Restriction 2

$$\forall\neg(road(cityA, cityB, length1) \wedge road(cityA, cityB, length2) \wedge abs(length1 - length2) > 10)$$

Restriction 3

$$\forall\neg(train(x, y, z) \wedge train(x', y', z') \wedge train(x'', y'', z'') \wedge x = x' \wedge y' = x'' \wedge y = y'' \wedge z < z')$$
$$\forall\neg(train(x, y, z) \wedge train(x', y', z') \wedge train(x'', y'', z'') \wedge x = x' \wedge y' = x'' \wedge y = y'' \wedge z < z'')$$