Name

CWID

Homework Assignment 3

April 2015

CS520 Results

Please leave this empty!

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
- In this assignment we will work with some existing tools to get experience in using the matching and mapping techniques we have discussed in class.
- We first give an overview of how to get these tools working on your machine and then discuss what tasks you should perform with these tools.

Coma 3.0

- *Coma 3.0* is a tool for schema matching that is freely available at `http://dbs.uni-leipzig.de/Research/coma.html`. It is implemented in Java
- The system requires a running MySQL server on your machine and will not start unless it is able to connect to this MySQL server
- Configuration settings are in `coma.properties`
  - You can set the JDBC connection URL as well as user and password for your MySQL server instance there
- The program comes with a `coma.bat` to start it on windows. On unix or Mac OS you need to write a shell script or call `java` directly to run it. See below for an example script.

```
export CLASSPATH="lib/coma-gui.jar:lib/coma-engine.jar\
:lib/additional/*:lib/maven/*:lib/additional/*"
java -cp ${CLASSPATH} -Xmx500M de.wdilab.coma.gui.Main
```

++Spicy

- *++Spicy* is a data exchange tool implemented in Java.
- You can download it for free from `http://www.db.unibas.it/projects/spicy/`
- It can use a backend Postgres or DB2 database or also works without a database
- The website also has several examples, e.g., `http://www.db.unibas.it/projects/spicy/software/examples.zip`

Part 2.1 Coma 3.0: (Total: 0 Points)

- Coma supports several file formats, e.g., XML schema. For instance, you can use some of the XML schemas from the ++spicy examples
- Load such files as source and target and run the matchers

Part 2.2 ++spicy: Use Examples to Understand the Tool (Total: 0 Points)

- Install ++spicy
- Download the examples mentioned above
- Startup the ++spicy GUI
- You can load existing scenarios from the examples files
- ++spicy supports all tasks needed for data exchange
  - schema matching
  - mapping generation
  - creating transformations
  - executing transformations
- test mapping generation
  - load the `personCarCity` example
  - run `generate transformations` from the `map` menu item. This creates mappings
  - inspect some of the mappings that were generated, observe how they use schema matches
- test creating transformations
  - run `generate SQL` from the `map` menu item.
  - inspect the code

Theory Part

- This part of the assignment helps you to practice the techniques we have introduced in class.

Part 2.3 Schema Matching (Total: 0 Points)

Source	Target
Company	**Brand**
CompName	BrandName
StockPrice	Headquarter
NumExpl	StockValue
Model	**Car**
ModelNumber	Color
Color	ModelSerial

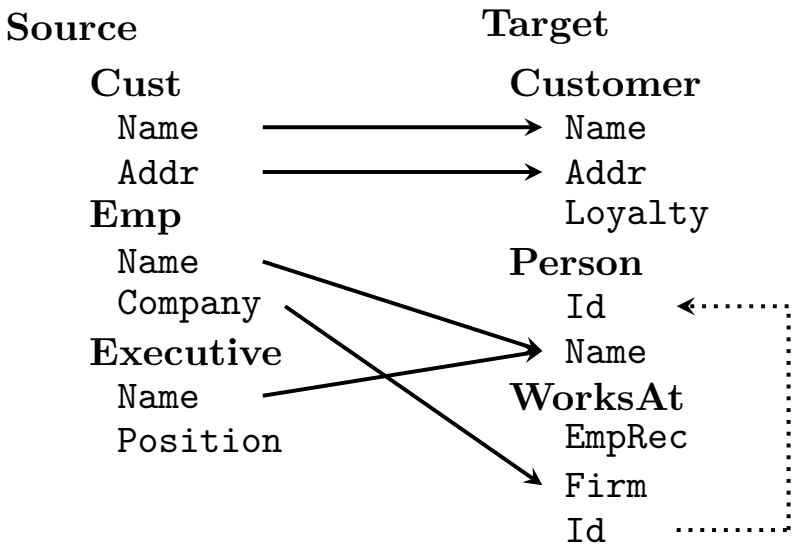
Question 2.3.1 Element Name Matching (0 Points)

Consider the two schemas above and compute similarity of schema elements using a matcher that uses edit distance between element names (attributes) as similarity measure. Recall that a similarity-based matcher has to compute all pairwise similarities and uses a threshold to determine matches. Compute all pairwise similarities and consider all pairs with edit distance less than < 7 as matches

Solution

	BrandName	Headquarter	StockValue	Color	ModelSerial
CompName	**5**	9	7	**6**	10
StockPrice	9	10	**4**	8	9
NumExpl	9	10	9	7	10
ModelNumber	8	9	10	8	**6**
Color	9	10	8	**0**	8

Part 2.4 Schema Mappings (Total: 0 Points)



Question 2.4.1 (0 Points)

Consider the source and target schemas shown above. Arrows form source attributes to target attributes represent schema matches. The dotted arrows represent foreign key constraints. Use the Clio algorithm presented in the data exchange part of class to 1) determine all source and target associations and 2) construct schema mappings using these associations and the schema matches (attribute correspondences). Recall that 1) is solved by chasing the inclusion constraints (foreign keys) and then removing associations that are subsumed by other associations. Write down the associations first and then the mappings as source-to-target tuple-generating dependencies (st-tgds). For the second step, recall that mappings are generated by combining a source with a target association and all correspondences that are covered by these two associations. Note that Clio would remove subsumed mappings. For sake of this homework do not remove such mappings, but mark them in the result. A pair of associations is subsumed by another pair of associations if both cover the same correspondences and the relations used by the subsumed pair are a superset of the relations covered by the other pair.

Solution

Since there are no foreign keys all source relations are single relation. Source associations are:

$$Cust(X_1, X_2)$$
$$Emp(X_1, X_2)$$
$$Executive(X_1, X_2)$$

Target associations are:

$$Customer(Y_1, Y_2, Y_3)$$
$$Person(Y_1, Y_2)$$
$$WorksAt(Y_1, Y_2, Y_3), Person(Y_3, Y_4)$$

Based on this we get 9 possible combinations of source with target associations, but not all of them cover any correspondences. Recall that mappings are generated by a direct translation of the associations into the LHS and RHS of a st-tgd. Variables in the RHS are replace by variables in the LHS if there exists an schema match (correspondence) between the corresponding source and target attributes. The resulting mappings are:

$$m_1 : \forall x, y : Cust(x, y) \rightarrow \exists z : Customer(x, y, z)$$
$$m_2 : \forall x, y : Emp(x, y) \rightarrow \exists z, a : Person(z, x) \wedge WorksAt(a, y, z)$$
$$m_3 : \forall x, y : Emp(x, y) \rightarrow \exists z : Person(z, x)$$
$$m_4 : \forall x, y : Executive(x, y) \rightarrow \exists z : Person(z, x)$$
$$m_5 : \forall x, y : Executive(x, y) \rightarrow \exists z, a, b : Person(z, x) \wedge WorksAt(a, b, z)$$

Mapping m_5 is subsumed by mapping m_4 because the pairs of associations that were used to generate these mappings cover the same correspondences (the correspondence from `Executive.Name` to `Person.Name`) and the relations covered by m_5 are a superset of the relations covered by mapping m_4.

Part 2.5 Virtual Data Integration (Total: 0 Points)

Question 2.5.1 (0 Points)

Use the mappings from the previous part (recall the st-tgds are equivalent to GLAV mappings under the open world assumption). As a first step write down the GLAV expressions equivalent to these st-tgs.

Find maximally contained rewritings for the queries shown below. Remember that queries are rewritten using GLAV mappings by first finding a rewriting using the views over the global (target) schema using any of the algorithms for maximally contained rewritings introduced in class, then replace the global schema views with the source schema views and unfold them.

Use the bucket algorithm for the first step. Recall that the bucket algorithm groups views based on which query predicates they cover, builds all combinations of picking one view per bucket, and does a query containment check to determine whether the generated rewriting is contained in the query. In case a candidate is not contained then the algorithm will try whether it is possible to add predicates to the candidate to make it contained (if we do not consider queries and views with contained predicates then this is limited to equating variables). All contained rewritings are then combined using union to create the maximally contained rewriting for the query.

$$Q_1(X) :- Person(Y, X)$$
$$Q_2(X, Y) :- Person(Z, X), WorksAt(A, Y, Z)$$

Solution

The GLAV versions of the mappings are

$$Q_1^G(X, Y) :- Customer(X, Y, Z) \supseteq Q(X, Y)_1^L :- Cust(X, Y)$$
$$Q_2^G(X, Y) :- Person(Z, X), WorksAt(A, Y, Z) \supseteq Q(X, Y)_2^L :- Emp(X, Y)$$
$$Q_3^G(X) :- Person(Z, X) \supseteq Q(X)_3^L :- Emp(X, Y)$$
$$Q_4^G(X) :- Person(Z, X) \supseteq Q(X)_4^L :- Executive(X, Y)$$

Recall that to rewrite a query using GLAV mappings we

- use any algorithm for finding a maximally contained rewriting (in our case the bucket algorithm) using only the global views, e.g., Q_2^G.
- in the rewriting replace the global with the corresponding local views, e.g., Q_2^G would be replaced with Q_2^L
- unfold the local views

Solution

Rewriting Q_1

This query has one atom, so we create only one bucket for subgoal `Person` containing all views that fulfill the 3 conditions that the bucket algorithm checks before inserting a view into a bucket. These are:

- The view body contains the relation of the subgoal
- The combination of the query's interpreted predicates (comparisons) and view's interpreted predicates should be satisfiable
- If the subgoal contains a variable from the head of the query, then the corresponding variable in the view subgoal should also be in the head of the view. For such variables and all variables that occur in the subgoal as well as the view head, we replace the view variable with the query variable. All remaining variables in the view head are replaced with fresh variables that occur neither in the view nor the query.

These are Q_2^G, Q_3^G, and Q_4^G (each view has a subgoal `Person`, the view variable corresponding to query head variable X occurs in the view head too, and there are no interpreted predicates). The content of the single bucket is:

$$\{Q_2^G(X, Y'), Q_3^G(X), Q_4^G(X)\}$$

To understand why variable in X is not renamed in all cases, but Y is renamed Q_1' recall that a variable in a bucket subgoal is replaced with the corresponding variable in the query if the variable is in the query head and view head. We keep the view variable if it does not occur in the query head, but in the view head. We replace all remaining view head variables with fresh variables. Thus, Y in the head of view Q_2^G does not appear in the bucket goal and is replaced with a new variable (here we chose Y').
We generate three potential rewritings:

$$Q_1'(X) = Q_2^G(X, Y')$$
$$Q_1''(X) = Q_3^G(X)$$
$$Q_1'''(X) = Q_4^G(X)$$

Solution

Next we have to check for each of these rewritings whether it is contained in the query (or can be made contained by adding additional comparison atoms). For that we have to unfold the views and apply a standard query containment check (finding a containment mapping). It turns out that all three potential rewritings are contained in the query.
For instance, consider

$$Q_1'(X) = Person(Z, X), WorksAt(A, Y', Z)$$

A valid containment mappings from $Q_1 \to Q_1'$ is $X \to X$ and $Y \to Z$.
Now, we can replace the global schema views with the source schema views and unfold them:

$$Q_1'(X) :- Emp(X, Y')$$
$$Q_1''(X) :- Emp(X, Y)$$
$$Q_1'''(X) :- Executive(X, Y)$$

Since the first two queries are the same modulo variable renaming, this gives us a maximally contained rewriting of the query:

$$Q_1(X) :- Emp(X, Y)$$
$$Q_1(X) :- Executive(X, Y)$$

Solution

Rewriting Q_2

This query has two atoms, so we create two buckets: one containing all views that for relation `Person`, and another containing all views for relation `WorksAt`.

Checking the 3 conditions we get the buckets:

$$Person : Q_2^G(X, Y'), Q_3^G(X), Q_4^G(X)$$
$$WorksAt : Q_2^G(X', Y)$$

Thus, we generate three potential initial rewritings.

$$Q_2'(X, Y) : -Q_2^G(X, Y'), Q_2^G(X', Y)$$
$$Q_2''(X, Y) : -Q_3^G(X), Q_2^G(X, Y)$$
$$Q_2'''(X, Y) : -Q_4^G(X), Q_2^G(X, Y)$$

Q_2'' and Q_2''' after unfolding are contained in Q. Rewriting Q_2' is not contained in the query Q. We have to either equate X with X' or to equate Y' and Y. In both cases we get one atom $Q_2^G(X, Y)$ and a second atom that is redundant and can be removed, giving us:

$$Q_2'(X, Y) : -Q_2^G(X, Y)$$
$$Q_2''(X, Y) : -Q_3^G(X), Q_2^G(X, Y)$$
$$Q_2'''(X, Y) : -Q_4^G(X), Q_2^G(X, Y)$$

Replace the global schema views with the source schema views and unfold them and get the maximally contained rewriting

$$Q_2(X, Y) : -Emp(X, Y)Q_2(X, Y) \quad : -Emp(X, Z), Emp(X, Y)Q_2(X, Y) : -Executive(X, Z), Emp(X, Y)$$

If we like to we can also eliminate the last two rules since they are contained in the first one.

$$Q_2(X, Y) : -Emp(X, Y)$$

Part 2.6 Data Exchange (Total: 0 Points)

Consider the following transportation database schema and example instance:

Cust

Name	Addr
Peter	Chicago
Bob	South Park
Alice	Chicago

Emp

Name	Company
Gert	IBM
Pferd	Oracle

Executive

Name	Position
Pferdegert	Senior VP
Heinzgert	CPO

Question 2.6.1 (0 Points)

Consider the source instance for the schema from the previous questions shown above. Use the data exchange query generation technique discussed in class to generate SQL queries implementing the mappings designed in the previous questions. Show the result of running these queries over the example source instance.

Solution

Recall that queries are generated by putting the LHS of a st-tgd into a FROM claus and one of the RHS atoms as a select clause. Such fragments for all st-tgds having a target relation R in their RHS are connected by union. For any existentially quantified variable we create a new skolem function using the graph algorithm presented in the slides to determine the arguments for this skolem function.
Recall that the algorithm for determining arguments works as follows. We

- Create a node for each target relation and each target relation attribute. We only create nodes for source attributes if their values are copied to the target (we use a variable in this position that occurs in the RHS of the st-tgd)
- We create an edges between target relations and their attributes. Two target attributes are connected if they use the same variable. A source and a target attribute is connected if they use the same variable.
- We annotate each target attribute connected to a source attribute with that source attribute
- We then propagate annotations according to the following rules
  - Propagate annotations from attributes to relations
  - Propagate annotations from relations to attributes (Only if attribute uses existentially quantified variable)
  - Propagate annotations between target attributes connected by equality edges
- The final annotations of a target attribute are the arguments to the skolem function

Customer:

We get the following query for the st-tgd mapping to relation *Customer* in our example:

$$m_1 : \forall x, y : Cust(x, y) \rightarrow \exists z : Customer(x, y, z)$$

```
SELECT Name, Addr, 'SK1(' || Name || ',' || Addr || ')' AS Loyalty
FROM Cust
```

Person:

We get the following query for the three st-tgds mapping to relation *Person* in our example:

$$m_2 : \forall x, y : Emp(x, y) \rightarrow \exists z, a : Person(z, x) \wedge WorksAt(a, y, z)$$
$$m_3 : \forall x, y : Emp(x, y) \rightarrow \exists z : Person(z, x)$$
$$m_4 : \forall x, y : Executive(x, y) \rightarrow \exists z : Person(z, x)$$

```
SELECT 'SK2(' || Name || ',' || Company || ')' AS Id, Name
FROM Emp
UNION
SELECT 'SK3(' || Name || ')' AS Id, Name
FROM Emp
UNION
SELECT 'SK4(' || Name || ')' AS Id, Name
FROM Executive
```

Solution

WorksAt:

For the only st-tgd mapping to relation *WorksAt* we get

$$m_2 : \forall x, y : Emp(x, y) \rightarrow \exists z, a : Person(z, x) \wedge WorksAt(a, y, z)$$

```
SELECT 'SK5(' || Name || ',' || Company || ')' AS EmpRec,
       Company AS Firm,
       'SK2(' || Name || ',' || Company || ')' AS Id
FROM Emp
```

Target Instance:

Running this queries over the provided source instance we get:

Customer

Name	Addr	Loyalty
Peter	Chicago	SK1(Peter,Chicago)
Bob	South Park	SK1(Bob,South Park)
Alice	Chicago	SK1(Alice,Chicago)

Person

Id	Name
SK2(Gert,IBM)	Gert
SK2(Pferd,Oracle)	Pferd
SK3(Gert)	Gert
SK3(Pferd)	Pferd
SK4(Pferdegert)	Pferdegert
SK4(Heinzgert)	Heinzgert

WorksAt

EmpRec	Firm	Id
SK5(Gert,IBM)	IBM	SK2(Gert,IBM)
SK5(Pferd,Oracle)	Oracle	SK2(Pferd,Oracle)