MAGIK: Managing Completeness of Data*

Ognjen Savković, Mirza Paramita, Sergey Paramonov, and Werner Nutt
Free University of Bozen-Bolzano
Piazza Domenicani 3, I-39100 Bolzano, Italy
{savkovic,nutt}@inf.unibz.it, {mparamita, sparamonov}@stud-inf.unibz.it

ABSTRACT

MAGIK demonstrates how to use meta-information about the completeness of a database to assess the quality of the answers returned by a query. The system holds so-called table-completeness (TC) statements, by which one can express that a table is partially complete, that is, it contains all facts about some aspect of the domain.

Given a query, MAGIK determines from such meta-information whether the database contains sufficient data for the query answer to be complete. If, according to the TC statements, the database content is not sufficient for a complete answer, MAGIK explains which further TC statements are needed to guarantee completeness.

MAGIK extends and complements theoretical work on modeling and reasoning about data completeness by providing the first implementation of a reasoner. The reasoner operates by translating completeness reasoning tasks into logic programs, which are executed by an answer set engine.

Categories and Subject Descriptors

H.2.7 [**Database Management**]: Database Administration—*Repository, Data warehouse, Security, integrity, and protection*

General Terms

Algorithms, Management

Keywords

Data Quality, Data Completeness, Answer Set Programming

1. INTRODUCTION

Completeness is a central aspect of data quality which only recently has received increased attention in research (cf. [1, 2]) In particular, there are no established techniques and systems to manage the completeness of data sets up to now.

Recently, Razniewski and Nutt revisited the problem of data completeness. The work was motivated by a project to create a school information system that can give guarantees about the completeness of its data—an application that shares characteristics with other applications in data integration and decision support [6]. In this scenario, a database instance D may be *partially complete* in that it contains, e.g., all pupils of classes 1a and 1b, but *generally incomplete* because it contains only some of the pupils of other classes.

Building upon previous work by Motro [5] and Levy [4], they developed a framework to reason about the question whether a generally incomplete database D contains sufficient information to return a complete answer for a specific query Q. In this framework, one can express which parts of which tables are complete, using so-called table-completeness (TC) statements (also called "local completeness statements" in [4]). For instance, one can write TC statements which say that the database instance contains "all pupils of class 1a" (TC_{1a}), "all pupils of class 1b" (TC_{1b}), "all pupils in classes of the science branch" (TC_{sc}) or the "humanities branch" (TC_{hum}) of a school, respectively. We expect that in an application, TC-statements will be generated automatically. For instance, a workflow engine could create a statement after concluding the task of registering the pupils of a class.

Given a collection of TC statements that hold over an instance D of our school database, we can ask whether they entail that for a certain query Q the set of answers $Q(D)$ is complete, that is, whether $Q(D)$ contains all answer records that one would expect if D were complete. Examples of queries are the ones for "all pupils in a class of level 1" (Q_{lev1}), or "all pupils of the school" (Q_{sch}).

To find out whether a set of TC statements implies completeness of a query is called *TC-QC reasoning.* In [6], the authors assessed the complexity of TC-QC reasoning for several types of TC statements and queries, under both set and bag semantics.

They did not show, however, how to implement reasoners for these problems. Neither did they take into account the integrity constraints that hold over a database, although in many cases query completeness is only implied by a set of TC statements *together* with key, foreign key (FK), or finite domain constraints (FD). For instance, in general completeness of the query Q_{lev1} does not follow from TC_{1a} and TC_{1b} alone, but it does if we know in addition that the only code letters for a class are "a" and "b", which is a finite domain constraint. As another example, consider query Q_{sch} and the two statements TC_{sc} and TC_{hum}. Then completeness of Q_{sch} does not follow from these two TC statements alone, but it does if in addition there is the FK constraint that every pupil belongs to a class in some branch *and* if there is the FD constraint that science and humanities are the only branches.

With the MAGIK demonstrator for completeness reasoning, we have made the following contributions:

- We have realized the first system that can check whether a set of table completeness statements $\mathcal{C}$ entails completeness of a query Q, as defined in [4, 6], both under set and bag semantics, provided $\mathcal{C}$ and Q are expressed by conjunctive queries without built-in predicates.
- We have gone beyond [6], by enabling MAGIK to take into account finite domain and (acyclic) foreign key constraints.

*This work was partially supported by the ESF project 2-299-2010 "SIS–Wir verbinden Menschen", and the European commission FP7 project ICT-2009.4.2 "TERENCE".

CIKM'12, October 29–November 2, 2012, Maui, HI, USA.
ACM 978-1-4503-1156-4/12/10.

- We have developed a technique to leverage answer-set programming for completeness reasoning. We translate a TC-QC inference task into a logic program whose answer sets have to have a certain property for the inference to be valid.
- We have also developed a component for explanations and suggestions.

Section 2 presents a possible demo session of MAGIK, Section 3 sketches the translation of completeness reasoning to answer set programming, and Section 4 gives an overview of the architecture.

2. SAMPLE DEMONSTRATION

MAGIK can reason about queries posed over a fixed schema with key, FK, and FD constraints. Schema and constraints can be loaded from an existing database or can be manually edited. One interacts with MAGIK via a web interface.

This section shows by way of examples which kinds of reasoning problems MAGIK solves and how one can interact with the system. We assume that we have defined the schema of a toy school database with the relations

$$\texttt{pupil}(\underline{\texttt{name}}, \texttt{level}, \texttt{code})$$
$$\texttt{class}(\underline{\texttt{level}}, \underline{\texttt{code}}, \texttt{branch})$$
$$\texttt{learns}(\underline{\texttt{name}}, \underline{\texttt{lang}}).$$

Here, $\texttt{pupil}(\texttt{fred}, 1, \texttt{a})$ means that Fred is a pupil in class 1a; $\texttt{class}(1, \texttt{a}, \texttt{sci})$ means class 1a belongs to the science branch; and $\texttt{learns}(\texttt{fred}, \texttt{french})$ that Fred learns French. Underlined attributes make up the primary keys and foreign key constraints hold as one would expect.

Figure 2 is a screenshot of the main page of the MAGIK demo, where we have loaded the schema above as a *virtual schema,* that is, as a manually edited schema without connection to a database. The page contains four main components: (1) Foreign key constraints, (2) Finite domain constraints, (3) Table completeness statements, and (4) Queries. For each component one can create new entries and modify existing ones. To specify a reasoning task, one activates constraints, TC statements, and a query by clicking. Below, we discuss four examples, where MAGIK analyzes a query with respect to some constraints and TC statements.

Query Q_{1a}: "Select the names of all pupils in class 1a":

```
SELECT p.name
FROM   pupil as p
WHERE  p.level = '1' AND p.code = 'a'
```

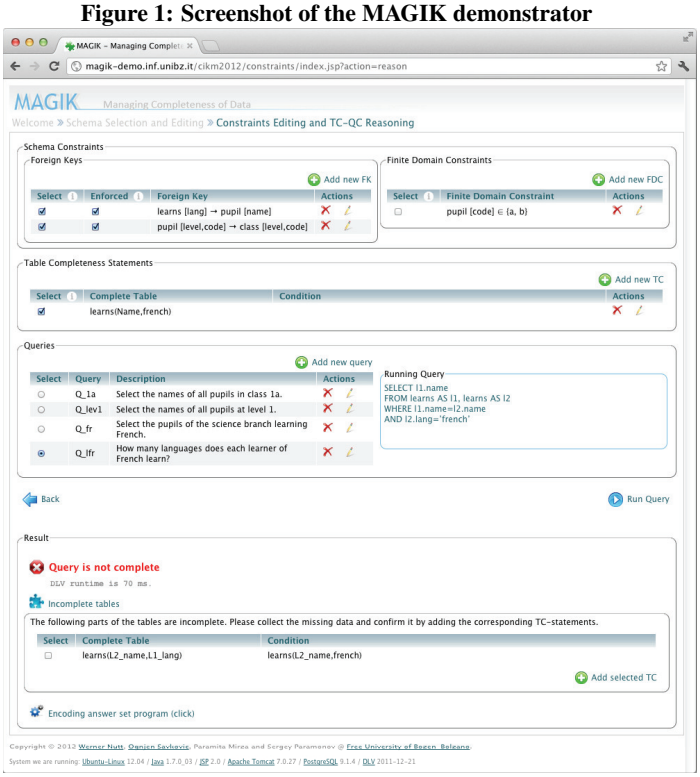
Suppose, we activate no constraints and no TC statements, only query Q_{1a}. By pushing "Run Query", we call the reasoner, which replies "Query is not complete". It analyzes, which parts of the database that are needed for the query are incomplete, suggests to supply the necessary data, and to confirm this by adding a TC statement. TC statements are written in a datalog-like syntax. Syntactically, a statement for a table R consists of two parts: an R-atom, representing a selection on R and, possibly, a condition, representing a semijoin with other tables. In the current example, the proposed statement has the form

Table: `pupil(P_name,1,a)` Condition:

The atom `pupil(P_name,1,a)`, where `P_name` is a variable and `1`, `a` are constants, represents all records satisfying the selection $\sigma_{\texttt{level=1,code=a}}(\texttt{pupil})$. We see that, unsurprisingly, MAGIK suggests the statement TC_{1a} from above, or, in words, it proposes to complete the pupils of class 1a.

Query Q_{lev1}: "Select the names of all pupils at level 1," obtained from Q_{1a} by dropping the condition "`p.code = 'a'`". We also

Figure 1: Screenshot of the MAGIK demonstrator



activate the FD constraint

$$\texttt{pupil[code]} \in \{\texttt{a}, \texttt{b}\},$$

which states that possible codes for classes are only `a` and `b`. Finally, we activate TC statement TC_{1a}. Clearly, Q_{lev1} cannot be answered completely because information about class 1a is complete, but not about class 1b. Exploiting the FD constraints, MAGIK is able to deduce this and suggests

Table: `pupil(P_name,1,b)` Condition:

that is, to complete the pupils of class 1b.

Query Q_{fr}: "Select the pupils of the science branch learning French":

```
SELECT p.name
FROM   pupil AS p, learns AS l, class AS c
WHERE  p.level = c.level AND p.code = c.code AND
       c.branch = 'science' AND
       l.name = p.name AND l.lang = 'french'
```

This example illustrates the role of foreign keys. We assume that our database is complete for all learners of French (TC_{fr}):

Table: `learns(Name,french)` Condition:

and that the FKs from `learns` to `pupil` and from `pupil` to `class` are enforced in our database. Then MAGIK reasons that for every learner of French, there are a corresponding `pupil` record and a corresponding `class` record. Thus, one can check for each learner of French whether they attend a class of the science branch, so that the query can be answered completely.

Query Q_{lfr}: "How many languages does each learner of French learn?":

```
SELECT l1.Name
FROM   learns AS l1, learns AS l2
WHERE  l1.Name = l2.Name AND
       l1.Lang = 'french'
```

Since bag semantics is the default in SQL, the query returns each name of a learner of French as many times as there are `learns` tuples with that name. If we only activate TC_{fr}, that is, if we suppose that we are complete for the French learners, but not necessarily for other languages, then MAGIK requests

Table: `learns(Tl2_name,Tl2_lang)`
Condition: `learns(Tl2_name,french)`,

that is, to complete the `learns` tuples for the names of those pupils who learn French (note, the request is not for all tuples, but only those whose `Tl2_name` variable satisfies the condition). If we add the keyword `DISTINCT`, that is, indicate that the query is to be evaluated under set semantics, then MAGIK finds out correctly that the query is complete, because tuple `l2` in the query is superfluous.

3. TRANSLATION INTO ANSWER SET PROGRAMMING

To check completeness of a query, MAGIK generates a disjunctive logic program, which is executed by the DLV answer set engine [3]. In essence, the program contains a protoypical complete database (the *ideal* database), where the query returns an answer. This ideal database is extended to satisfy FK and instantiated to satisfy FD constraints. The program then constructs for each instantiation a minimal *real* database, which contains just the necessary data that it is required to contain according to the TC statements. Finally, it tests whether the query also returns an answer over all the new *real* databases. If it does, then MAGIK replies "Query complete." If not, we generate an explanation out of the comparison between ideal and real database.

In our translation, we exploit various features of disjunctive logic programs and answer set programming (ASP): (1) an FK constraint is translated into a non-monotonic rule with a Skolem function, which generates a referenced tuple if none exists; (2) an FD constraint is translated into a disjunctive rule, which nondeterministically instantiates constrained variables with FD values; (3) cautious reasoning [3] is applied to check whether the query is satisfied by all resulting real databases.

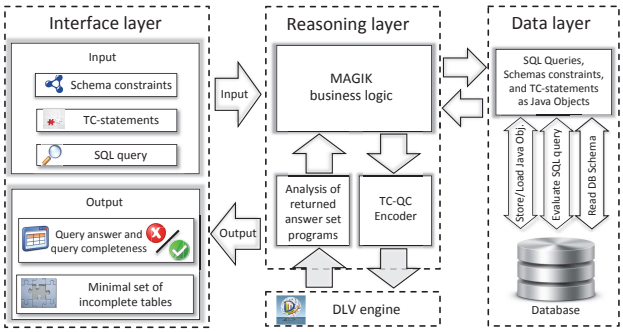
Our translation creates always programs that are just as complex as needed: for classes of reasoning problems that are in PTIME, in NP, or in Π_2^P, the resulting ASP programs have the same difficulty.

4. SYSTEM ARCHITECTURE, FUNCTIONALITIES AND USAGE

System architecture. MAGIK is a web application that consists of three layers (Figure 2). The *web interface layer* is implemented using Java Server Pages that are executed on an Apache-Tomcat Web server. The *reasoning layer* is the core part of the system. It encodes the the problem into a logic program and passes it on to the DLV answer set engine [3]. Based on the returned answers, it suggests the minimal set of TC statements that are missing to guarantee query completeness. Lastly, the *data layer* manages TC statements, queries and schema constraints. It can also connect to a database, extract schema and constraints, and evaluate the analyzed query. Currently, MAGIK is set up to communicate with PostgreSQL databases.

System functionalities. MAGIK can be accessed in two modes. The *database mode* illustrates a scenario where a completeness manager is added to an existing database. Here, MAGIK imports table declarations and foreign keys from the database catalog. In the *virtual mode* one can define arbitrary new schemas with keys and foreign keys for testing purposes. In both modes, FD constraints, TC statements and queries are created, edited, or deleted

Figure 2: Architecture of the MAGIK demonstrator



by the user. Currently, MAGIK accepts SQL select-project-join queries where selections are equality tests. Moreover, queries can have the key word `DISTINCT` or grouping with the aggregate function `count(*)`. MAGIK analyzes a query with respect to completeness and, in database mode, also evaluates the query.

System usage. After starting the demo the user is located on the schema selection page where several virtual and database schemas are offered. Then he chooses the mode and a schema and is led to the main page where he can create, edit, and activate the input parameters: FK constraints, FD constraints, TC statements and a query. When he runs the query, the result field appears and displays a "completeness certificate" for the query. Depending of the mode, also the query answer is printed. In case the query is not found to be complete, the system proposes a minimal set of TC statements that are needed to guarantee completeness of the query. Those TC statements can be added to the existing statements by clicking. Finally, one can inspect the answer set program that encodes the reasoning problem.

5. CONCLUSION

With MAGIK, we show how one could build a completeness management component for an information systems that collects data coming from different sources and/or different business processes and therefore is generally incomplete. MAGIK demonstrates the key functionalities of a such a component: it would document the content of an information system in terms of table completeness (TC) statements and it would analyze whether a query can be answered completely, based on the TC statements and the integrity constraints that the database satisfies. Such reasoning services have been proposed in the literature and have independently been requested by practitioners. MAGIK shows how such services can be realized with an answer set engine and offers a platform to experiment with them. The preview of the MAGIK demo is available at `http://magik-demo.inf.unibz.it/cikm2012`.

6. REFERENCES

[1] W. Fan and F. Geerts. Relative information completeness. In *PODS*, pages 97–106, 2009.

[2] W. Fan and F. Geerts. Capturing missing tuples and missing values. In *PODS*, pages 169–178, 2010.

[3] N. Leone, G. Pfeifer, W. Faber, T. Eiter, G. Gottlob, S. Perri, and F. Scarcello. The DLV system for knowledge representation and reasoning. *ACM TOCL*, 7(3):499–562, 2006.

[4] A. Levy. Obtaining complete answers from incomplete databases. In *Proc. VLDB*, pages 402–412, 1996.

[5] A. Motro. Integrity = Validity + Completeness. *ACM TODS*, 14(4):480–502, 1989.

[6] S. Razniewski and W. Nutt. Completeness of queries over incomplete databases. In *VLDB*, 2011.