Implementing Query Completeness Reasoning

Werner Nutt
Free Univ. of Bozen-Bolzano
Bozen, Italy
werner.nutt@unibz.it

Sergey Paramonov
KU Leuven
Leuven, Belgium
sergey.paramonov@kuleuven.be

Ognjen Savković
Free Univ. of Bozen-Bolzano
Bozen, Italy
ognjen.savkovic@unibz.it

ABSTRACT

Data completeness is commonly regarded as one of the key aspects of data quality. With this paper we make two main contributions: (i) we develop techniques to reason about the completeness of a query answer over a partially complete database, taking into account constraints that hold over the database, and (ii) we implement them by an encoding into logic programming paradigms. As constraints we consider primary and foreign keys as well as finite domain constraints. In this way we can identify more situations in which a query is complete than was possible with previous work. For each combination of constraints, we establish characterizations of the completeness reasoning and we show how to translate them into logic programs. As a proof of concept we ran our encodings against test cases that capture characteristics of a real-world scenario.

Categories and Subject Descriptors

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

Keywords

Data Quality; Data Completeness; Answer Set Programming

1. INTRODUCTION

In database applications such as information integration and decision support, one is interested in data sets that are complete, in the sense that the data represent all relevant facts that hold in the real world. In many situations, though, it is only possible to guarantee partial completeness of the data, which means that for certain aspects of the application domain the data are complete, but not for others. In such a situation, one would like to know at least whether the available data are sufficient to answer a given query completely, that is, whether the answers to the query over the available data are the same as if the data set were complete [5, 6, 8, 9].

School Information System. A typical scenario is the management of administrative data that are contributed by people. An example is the management of school data in the province of Bolzano, which motivated the work reported here. Data in the school information system of the province are often incomplete because each school is individually responsible for inserting its data into the system and because for certain kinds of data the contribution is optional. Decision makers, however, need to know whether or not the statistics on which they base the allocation of resources to schools are derived from complete data. As often some parts of the data are complete, one would like to automatically provide guarantees that some query answers are guarantees and hint at those that need further checks.

Example. Consider the following example of a partially complete school database. Among other data, the database contains facts about pupils and classes, e.g., "John belongs to class *a1* at Newton school", $pupil(john, a1, newton)$, or "Class *a1* at Newton school is at level 1 and it belongs to the science branch", $class(a1, newton, 1, sci)$.

Assume we know that we have all pupils from class *a1* and all pupils from class *b2* at Newton school. Thus, the query that asks for "all pupils in class *a1* at Newton school", $Q_{\text{a1n}}(N) \leftarrow pupil(N, a1, newton)$, has a complete answer. However, if we consider the query that asks for "all science pupils at Newton school", $Q_{\text{sn}}(N) \leftarrow pupil(N, C, newton),\ class(C, newton, L, sci)$, then under the completeness assumptions above we cannot say whether the query answer of Q_{sn} is complete. Note that, in principle, there can be other classes different from *a1* and *b2*.

Databases typically come with integrity constraints such as primary and foreign keys. But there can be also other kinds of constraints that restrict the domains of attributes.

Example (cont'd). Continuing the example, assume that there is a constraint on the *class* relation, saying that "if a class at Newton school belongs to the science branch then that class has a class code among *a1* and *b2*", expressed with $class\{school = newton, branch = sci\}[ccode] = \{a1, b2\}$. We call this a conditional finite domain constraint.

Now we can reason in the following way. The query Q_{sn} selects Newton pupils from the science branch. Since the above constraint holds for the *class* relation, and *pupil* is joined with *class*, pupils in Q_{sn} must be either from class *a1* or *b2*. Considering that we are complete for *a1* and *b2*, as assumed above, we conclude that we are complete for all pupils selected by Q_{sn}. Assume further that there exists a foreign key from *pupil* to *class* in our database, $pupil[ccode, sname] \subseteq class[ccode, sname]$. Then we have all records of *class* that can be joined with *pupil*, i.e. we have all *class* records selected by Q_{sn}. Overall, assuming the constraint and the foreign key, we can conclude that the answer to Q_{sn} is complete.

Related Work. Motro [17] was the first to formalize incomplete databases and completeness of queries. Levy [15], in addition, in-

CIKM'15, October 19–23, 2015, Melbourne, Australia.

DOI: http://dx.doi.org/10.1145/2806416.2806439.

troduced a format for assertions that say which parts of a relational database are complete. We call these assertions *table completeness* or *TC* statements. He raised the problem to determine whether a set of such TC statements imply that some given query can be answered completely. Razniewski and Nutt [20] showed how to reduce this *completeness reasoning problem* to containment of conjunctive queries [4] and gave a comprehensive analysis of its complexity, considering several variants of queries and assertions. In particular, they showed that for the most expressive queries and assertions they considered, completeness reasoning is Π_2^P-complete. Such a degree of difficulty is reached, for instance, if queries and assertions are expressed by conjunctive queries and if finite-domain constraints hold over the database.

Contributions. In this work, we consider databases that are incomplete in that entire tuples are missing. We address the problem of completeness reasoning for conjunctive queries and for TC statements in the presence of constraints. We generalize the previously investigated problem by taking into consideration two kinds of constraints. The first are primary and foreign key constraints, the second one are a new kind of constraints, that we call *conditional finite domain constraints* (CFDCs), which restrict the domain of relation attributes depending on the values of other attributes. CFDCs are more expressive than classical *finite domains constraints* (FDCs). On the other hand, if we have a database instance at disposal, reasoning about completeness with a database instance is typically intractable [20], and therefore not feasible for large instances in realistic scenarios. In this respect, we expect that CFDCs offer a good trade-off that allows for feasible running times while still carrying more information than FDCs.

Further, we provide a new approach to completeness reasoning which gives rise to an implementation technique of the problem that was not obvious from previous work [19]. A demonstrator system, MAGIK, which realizes our approach and which is publicly accessible on the Web[1] has been implemented based on the techniques in this paper and was presented in [21] and [22].

Our approach consists of two steps. First, we develop theorems that provide a syntactic characterization of the reasoning tasks for each combination of constraints. Based on these, we develop encodings of the problems into logic programming formalisms, where disjunction occurs in the head of the rules that encode CFDCs. One can run such encodings using Answer Set Programming (ASP) solvers (or Prolog engines if there are no CFDCs) such that the resulting program under cautious reasoning entails a test fact if and only if the original problem has the answer "yes". While logic programming is conceptually close to the characterizations, they can in principle also be used to establish encodings into other paradigms.

In the paper we present some of the proofs. For more technical ones we only provide the intuition, due to the space limitations. Complete proofs can be found in [18].

We tested our encoding using two state-of-the-art ASP solvers, `dlv`[2] [14] and `clingo`[3] [10], and one Prolog implementation, `swi`[4]. The obtained results show that our encoding is able to process inputs that mirror the requirements of real world systems.

The remainder of the paper is organized as follows. In Section 2, we recall basic definitions from database theory and fix our notation. Section 3 formally introduces completeness reasoning. In Section 4, we characterize completeness reasoning in the absence of constraints and present our encoding for this case. Characterization and encoding are generalized in Section 5 to take account of key and foreign key constraints, and in Section 6 to conditional finite domain constraints into account, while the combination of both is studied in Section 7. Section 8 investigates the problem of finding a complete generalization of an incomplete query. Section 9 reports on our experiments and Section 10 concludes.

[1] `http://magik-demo.inf.unibz.it`

[2] `http://www.dlvsystem.com`

[3] `http://potassco.sourceforge.net`

[4] `http://www.swi-prolog.org`

2. PRELIMINARIES

Relational Databases and Conjunctive Queries. We assume an infinite set of constants, *dom*, and a database schema, Σ, with an infinite set of relations. Each relation is identified by a relation name, e.g., R, and a list of attributes, $att(R)$. We use $ary(R)$ to denote the arity of R. In the following, we assume the schema to be fixed. For a relation R with arity n, an *atom* is an expression $R(t_1,\ldots,t_n)$, where $t_1 \ldots t_n$ are either elements of *dom* or variables. We denote constants with lower-case and variables with upper-case letters. A database *instance* D is a finite set of ground atoms, that is, atoms that contain only constants. We sometimes refer to the atoms in an instance as facts. For a relation R, we denote as $R(D)$ the set $\{\bar{t} \mid R(\bar{t}) \in D\}$ of all tuples occurring in an R-atom in D. A *condition* is a set of atoms.

A *conjunctive query* is written as $Q(\bar{X}) \leftarrow B$, where B is a condition and $\bar{X}$ is a tuple of variables, each of which occurs also in B. We call B the *body* of Q, the variables in $\bar{X}$ the *distinguished variables* and the other variables in B the *nondistinguished variables*. Given a conjunctive query $Q(\bar{X}) \leftarrow B$ and an instance D, an answer to Q is a tuple $\alpha\bar{X}$, where α is an assignment of domain values to variables such that $\alpha B \subseteq D$. The *set* of all answers to Q over D is written as $Q^s(D)$. Similarly, we define $Q^b(D)$ as the *bag* of answers that contains as many copies of a tuple as there are assignments returning it. We say that Q is evaluated under *set* or *bag semantics,* respectively, if we refer to the set or bag of answers.

Constraints. Database systems allow one to formulate conditions, so-called (integrity) constraints, that all instances of a database have to satisfy. Many of them can be captured in logic and allow for more inferences when reasoning about instances and queries. In this paper, we consider key constraints, foreign key constraints, and conditional finite domain constraints.

A *primary key* (PK) constraint has the form $Key(R,A)$, where R is a relation symbol and A is a sublist of $att(R)$. A *foreign key* (FK) constraint has the form $R[A] \subseteq S[B]$, where R, S, are relations, A is a sublist of $att(R)$, and B is from $Key(S,B)$. Satisfaction of PK and FK constraints are defined as usual (cf. [1]).

In this paper, we also consider *weakly acyclic* sets of FK constraints, which have been introduced by Fagin et al. For the definition, we refer to their paper [7].

A *conditional finite domain* (CFD) constraint has the form $R\{A = \bar{v}\}[a] = W$, where R is a relation, A is a sublist of $att(R)$, $\bar{v}$ is a list of constants of the same length as A, a is an attribute from $att(R)$ that is not in A, and W is a set of constants. An instance D satisfies such a constraint if for every fact $R(\bar{t}) \in D$ we have that if $R(\bar{t})[A] = \bar{v}$ then $R(\bar{t})[a] \in W$, where $R(\bar{t})[A]$ is the projection of the values of $R(\bar{t})$ at the attribute places from A and $R(\bar{t})[a] \in W$ is defined similarly. Alternatively, if $A = a_1,\ldots,a_n$ and $\bar{v} = v_1,\ldots,v_n$, we write the condition $A = \bar{v}$ as a list of equalities, $a_1 = v_1,\ldots,a_n = v_n$. For example, the CFD constraint that states "a class at Newton school from the science branch has as code either *a1* or *b2*", is expressed as $class\{school = newton, branch = sci\}[ccode] = \{a1, b2\}$. If A and $\bar{v}$ are empty in a CFD constraint, we simply write $R[a] = W$ and call this a *finite domain* (FD) constraint.

We will use $\mathcal{K}$ and $\mathcal{F}$ generically to denote sets of FK, and CFD constraints, respectively.

Logic Programming. A *disjunctive rule* has the form

$$A_1 \mid \ldots \mid A_k \leftarrow B_1, \ldots, B_n,$$

where the A's and B's are atoms (see [11]). If $k = 1$, that is, the head consists of a single atom, we say that the rule is a *Horn* rule. A *fact* is a Horn rule with empty body (we omit "$\leftarrow$"). A rule with empty head (the empty disjunction) is a *denial*. (We avoid the common term "integrity constraint" in this paper to avoid confusion with integrity constraints over databases.) An *answer-set program* is a finite set of disjunctive rules. For the theory of answer set programs we refer to [23] and [3]. A *positive logic program* contains only Horn rules. Positive logic programs can be executed under SLD resolution as implemented in Prolog [2].

3. DATA COMPLETENESS

Running Example. Our examples build upon a toy schema from the school world with the three relations

$$\begin{aligned} &\mathit{pupil}(\underline{\mathit{name}}, \mathit{ccode}, \mathit{sname}),\\ &\mathit{class}(\underline{\mathit{ccode}}, \underline{\mathit{sname}}, \mathit{level}, \mathit{branch}),\\ &\mathit{takes}(\underline{\mathit{name}}, \underline{\mathit{activity}}). \end{aligned}$$

Here, $\mathit{pupil}(\mathit{fred}, \mathit{a1}, \mathit{newton})$ means that Fred is a pupil of class *a1* at Newton school; $\mathit{class}(\mathit{a1}, \mathit{newton}, 1, \mathit{sci})$ means that class *a1* at Newton school is a 1st level class that belongs to the science branch; and $\mathit{takes}(\mathit{fred}, \mathit{chess})$ means that Fred is taking chess as an extracurricular activity.

Underlined attributes indicate the primary key, which in our notation can be expressed as the key constraints $\mathit{Key}(\mathit{pupil}, \{\mathit{name}\})$, $\mathit{Key}(\mathit{class}, \{\mathit{ccode}, \mathit{sname}\})$, and $\mathit{Key}(\mathit{takes}, \{\mathit{name}, \mathit{activity}\})$. We also consider two FK constraints, which say that every class, identified by class code and school name, referred to by a *pupil* tuple occurs in the *class* relation, and every pupil name referred to by a *takes* tuple occurs in the *pupil* relation. Formally, this is expressed as $\mathit{pupil}[\mathit{ccode}, \mathit{sname}] \subseteq \mathit{class}[\mathit{ccode}, \mathit{sname}]$ and $\mathit{takes}[\mathit{name}] \subseteq \mathit{pupil}[\mathit{name}]$. Finally, we consider a CFD constraint that restricts classes at Newton school to belong either to the science or the humanities branch, expressed as $\mathit{class}\{\mathit{sname} = \mathit{newton}\}[\mathit{branch}] = \{\mathit{sci}, \mathit{hum}\}$.

The conjunctive query Q_{s1n}, defined by the rule

$$Q_{\mathrm{s1n}}(N) \leftarrow \mathit{pupil}(N, C, \mathit{newton}), \mathit{class}(C, \mathit{newton}, 1, \mathit{sci}) \qquad (1)$$

asks for "the names of all pupils from Newton school of the 1st level that attend a class from the science branch".

Query and Table Completeness. When stating that data is incomplete, one must have a conceptual complete reference. We model an *incomplete database* in the style of [17] as a pair of database instances $\mathcal{D} = (D^i, D^a)$, where $D^a \subseteq D^i$. Here, D^i is called the *ideal* state and D^a the *available* state. In an application, the state stored in a DBMS is the available state, which often represents only a part of the facts that hold in reality. The facts holding in reality constitute the ideal state, which however is unknown. (Later on we will introduce table completeness statements as a way to express meta-information about the extent to which the available state captures the ideal state.)

Data are accessed by posing queries. We would like to know whether a database has sufficient information to answer a query completely, that is, whether the query is *complete*. If we can infer from meta-information that a query is complete, we know that the answer we receive over the available database is the same as the one we would get over the (hypothetical) ideal database.

We write $\mathit{Compl}^s(Q)$ and $\mathit{Compl}^b(Q)$, respectively, to indicate that Q is complete under set or bag semantics. The first statement is satisfied by an incomplete database $\mathcal{D} = (D^i, D^a)$, written $\mathcal{D} \models \mathit{Compl}^s(Q)$, if $Q^s(D^i) = Q^s(D^a)$. Analogously, the second is satisfied if $Q^b(D^i) = Q^b(D^a)$.

With *table completeness* (TC) statements we specify that parts of a table are complete. A TC statement, written $\mathit{Compl}(R(\bar{s}); G)$, has two components, a relational atom $R(\bar{s})$ and a condition G. In the sequel, we will denote a TC statement generically as C. As an example, consider

$$C_{\mathrm{sci}} \;=\; \mathit{Compl}(\mathit{pupil}(N, C, S); \mathit{class}(C, S, L, \mathit{sci})) \qquad (2)$$

$$C_{\mathrm{lev1}} \;=\; \mathit{Compl}(\mathit{class}(C, S, 1, B); \mathit{true}). \qquad (3)$$

The first statement asserts that the table *pupil* contains all records of pupils from science classes, while the second asserts that the table *class* contains all records of classes at level 1.

To define the semantics of a TC statement C, we associate a query $Q_C(\bar{s}) \leftarrow R(\bar{s}), G$ to it. Then C is satisfied by $\mathcal{D} = (D^i, D^a)$, written $\mathcal{D} \models \mathit{Compl}(R(\bar{s}); G)$, if $Q_C(D^i) \subseteq R(D^a)$. This means that the ideal instance D^i is used to determine those tuples in the ideal version $R(D^i)$ that satisfy G, and that the statement C is satisfied if these tuples are present in the available version $R(D^a)$. We refer to the query associated to C as Q_C. For instance, the query associated to C_{sci} is $Q_{C_{\mathrm{sci}}}(N, C, S) \leftarrow \mathit{pupil}(N, C, S), \mathit{class}(C, S, L, \mathit{sci})$.

To reason about completeness, we define for every set $\mathcal{C}$ of TC statements the operator $T_{\mathcal{C}}$ that maps database instances to database instances. If C is a TC statement about R, then we define $T_C(D) := \{R(\bar{t}) \mid \bar{t} \in Q_C(D)\}$. For $\mathcal{C}$ we define

$$T_{\mathcal{C}}(D) := \bigcup_{C \in \mathcal{C}} T_C(D). \qquad (4)$$

The operator $T_{\mathcal{C}}$ is monotonic, and for every instance D, the pair $(D, T_{\mathcal{C}}(D))$ is an incomplete database satisfying $\mathcal{C}$, and $T_{\mathcal{C}}(D)$ is the smallest set (wrt set inclusion) for which this holds. We formalize this with the following claim that we need for the proofs later on.

PROPOSITION 1 ([20]). *Let $\mathcal{C}$ be a set of TC statements. Then*

a) $T_{\mathcal{C}}(D) \subseteq D$, for all database instances D;

b) $(D^i, D^a) \models \mathcal{C}$ iff $T_{\mathcal{C}}(D^i) \subseteq D^a$, for all $D^a \subseteq D^i$.

We say that $\mathcal{C}$ entails the completeness of Q with respect to set semantics, written $\mathcal{C} \models \mathit{Compl}^s(Q)$, if every incomplete database $\mathcal{D}$ that satisfies the statements in $\mathcal{C}$, also satisfies $\mathit{Compl}^s(Q)$. The meaning of "$\mathcal{C} \models \mathit{Compl}^b(Q)$" is defined analogously.

The *completeness reasoning problem* is to check, given $\mathcal{C}$ and Q, whether the entailments above hold. An instance of this problem for set semantics is to check whether $\{C_{\mathrm{sci}}, C_{\mathrm{lev1}}\} \models \mathit{Compl}^s(Q_{\mathrm{s1n}})$ (which intuitively holds and which we prove to hold in Section 4).

Let $\mathcal{F}$ be a set of CFD constraints and $\mathcal{K}$ be a set of key and FK constraints. We say that $\mathcal{D} = (D^i, D^a)$ *satisfies* $\mathcal{F}$ (resp., $\mathcal{K}$) if D^i satisfies $\mathcal{F}$ (resp., $\mathcal{K}$). Note that, due to $D^a \subseteq D^i$, this implies that also D^a satisfies $\mathcal{F}$ and that D^a satisfies the key constraints in $\mathcal{K}$. We say that $\mathcal{K}$ is *enforced* over $\mathcal{D}$ if both D^i and D^a satisfy $\mathcal{K}$. If $\mathcal{D}$ satisfies $\mathcal{K}$, this means that in the real world, represented by the ideal database, the FK constraints hold. If $\mathcal{K}$ is enforced over $\mathcal{D}$, then the constraints of $\mathcal{K}$ hold also over the available database. For instance, we may know that all pupils at our school belong to a class (because there are no external pupils), but the information about the levels and branches of classes contained in the relation class has not been entered into the available database. In that case, $\mathcal{D}$ satisfies $\mathit{pupil}[\mathit{ccode}, \mathit{sname}] \subseteq \mathit{class}[\mathit{ccode}, \mathit{sname}]$, but the constraint is not enforced.

We say that $\mathcal{C}$ entails the completeness of Q wrt $\mathcal{F}$, and write $\mathcal{C} \models_{\mathcal{F}} \mathit{Compl}^s(Q)$, if every $\mathcal{D}$ that satisfies both $\mathcal{C}$ and $\mathcal{F}$, also satisfies $\mathit{Compl}^s(Q)$. Analogously, we define "$\models_{\mathcal{K}}$", while "$\models^e_{\mathcal{K}}$" means that only such $\mathcal{D}$'s are considered where $\mathcal{K}$ is enforced.

From [20] we know that the completeness reasoning problem, both for set and bag semantics, can be reduced to query containment (cf. [4] and [13]). Reasoning in the presence of FD constraints can be reduced to containment with respect to FD constraints [19].

4. PLAIN COMPLETENESS REASONING

In previous work, Razniewski and Nutt reduced completeness reasoning to query containment by which they readily obtained complexity results [20]. However, the reduction did not give rise to an implementation technique. In this paper, we will reduce completeness checking to the question whether a test query returns a specific result over a test database. We will then use these characterizations to translate completeness checks into logic programs that can be executed by ASP or Prolog engines.

In the rest of the paper, we always consider a set of TC statements $\mathcal{C}$ and a conjunctive query Q defined by the rule $Q(\bar{X}) \leftarrow R_1(\bar{t}_1), \ldots, R_n(\bar{t}_n)$.

Characterizing Completeness Reasoning. While developing our approach, we illustrate it with an example. Consider the query Q_{s1n} in (1) and the set $\mathcal{C}_{\text{sci,lev1}} = \{C_{\text{sci}}, C_{\text{lev1}}\}$ comprising the TC statements in (2) and (3). Suppose $\mathcal{D} = (D^i, D^a)$ satisfies $\mathcal{C}_{\text{sci,lev1}}$. Consider an answer, say n', returned by Q_{s1n} over the ideal instance D^i. Then D^i contains two atoms of the form $\mathit{pupil}(n', c', \mathit{newton})$ and $\mathit{class}(c', \mathit{newton}, 1, \mathit{sci})$. Now, due to C_{sci}, also D^a contains $\mathit{pupil}(n', c', \mathit{newton})$, and due to C_{lev1}, the atom $\mathit{class}(c', \mathit{newton}, 1, \mathit{sci})$ is in D^a, too. Consequently, Q_{s1n} returns n' also over D^a. Since D^i and D^a were arbitrary, this shows that $\mathcal{C}_{\text{sci,lev1}} \models \mathit{Compl}^s(Q_{\text{s1n}})$.

Using the $T_{\mathcal{C}}$ transformation in (4), we can generalize this approach to a completeness test. We define the set of facts D_Q, which we call the *canonical database* of Q, obtained by freezing the atoms in the body of Q. ("Freezing" variables is a well-known concept in logic programming and database theory, which allows one to treat a variable like a constant.) Thus,

$$D_Q = \{R_1(\theta\bar{t}_1), \ldots, R_n(\theta\bar{t}_n)\},$$

where θ is the substitution that maps each variable X to the "frozen version" θX of X.

To check whether Q is complete we apply $T_{\mathcal{C}}$ to D_Q and check whether Q can retrieve the frozen tuple of distinguished variables.

THEOREM 2 (CHARACTERIZATION SET SEMANTICS). *Let $\mathcal{C}$ be a set of TC statements, and $Q(\bar{X}) \leftarrow B$ be a conjunctive query. Then*

$$\mathcal{C} \models \mathit{Compl}^s(Q) \iff \theta\bar{X} \in Q(T_{\mathcal{C}}(D_Q)).$$

PROOF. ($\Rightarrow$) Suppose that $\mathcal{C} \models \mathit{Compl}^s(Q)$. To show that $\theta\bar{X} \in Q(T_{\mathcal{C}}(D_Q))$, we construct the pair $\mathcal{D}_Q = (D_Q, T_{\mathcal{C}}(D_Q))$, which is an incomplete database according to Proposition 1 a). Obviously, it holds that $T_{\mathcal{C}}(D_Q) \subseteq T_{\mathcal{C}}(D_Q)$, and thus from Proposition 1 b) we conclude that $\mathcal{D}_Q \models \mathcal{C}$. Hence, $\mathcal{D}_Q \models \mathit{Compl}^s(Q)$ and $Q(D_Q) = Q(T_{\mathcal{C}}(D_Q))$. Further, D_Q is the frozen version of the body of Q, where the output variables $\bar{X}$ have been substituted with $\theta\bar{X}$. Thus $\theta\bar{X} \in Q(D_Q)$, and it follows that $\theta\bar{X} \in Q(T_{\mathcal{C}}(D_Q))$.

($\Leftarrow$) Suppose that $\theta\bar{X} \in Q(T_{\mathcal{C}}(D_Q))$. We want to show that $\mathcal{C} \models \mathit{Compl}^s(Q)$. To this end let $\mathcal{D} = (D^i, D^a)$ be an incomplete database such that $\mathcal{D} \models \mathcal{C}$. Suppose that $\bar{c} \in Q(D^i)$. We show that $\bar{c} \in Q(D^a)$. Let α be an assignment satisfying Q over D^i that retrieves $\bar{c}$. Then $\alpha\bar{X} = \bar{c}$ and $\alpha B \subseteq D^i$, where B is the body of Q. Since $\mathcal{C} \models \mathit{Compl}^s(Q)$, it follows that $T_{\mathcal{C}}(\alpha B) \subseteq D^a$. Now, let β be a satisfying assignment for Q over $T_{\mathcal{C}}(D_Q)$ that retrieves $\theta\bar{X}$. Then $\beta\bar{X} = \theta\bar{X}$ and $\beta B \subseteq T_{\mathcal{C}}(D_Q) = T_{\mathcal{C}}(\theta B)$. Consider the composed assignment $\alpha\theta^{-1}\beta$. Because of the inclusion just derived, it holds that $\alpha\theta^{-1}\beta B \subseteq \alpha\theta^{-1}T_{\mathcal{C}}(\theta B)$. It also holds that $\alpha\theta^{-1}T_{\mathcal{C}}(\theta B) \subseteq T_{\mathcal{C}}(\alpha\theta^{-1}\theta B)$, since α may map a variable to a constant appearing in $\mathcal{C}$, or may map distinct variables to the same constant, thus enabling more TC statements in $\mathcal{C}$ to be applied. Moreover, we have that $T_{\mathcal{C}}(\alpha\theta^{-1}\theta B) = T_{\mathcal{C}}(\alpha B) \subseteq D^a$, where the last inclusion has been derived above. In summary, we have shown that $\alpha\theta^{-1}\beta$ satisfies Q over D^a. We verify that $\bar{c} = \alpha\bar{X} = \alpha\theta^{-1}\theta\bar{X} = \alpha\theta^{-1}\beta\bar{X}$, which implies that $\bar{c}$ is the result retrieved by Q over D^a with the assignment $\alpha\theta^{-1}\beta$, so that $\bar{c} \in Q(D^a)$. □

Encoding Completeness Reasoning. For any Q and $\mathcal{C}$, the check whether $\theta\bar{X} \in Q(T_{\mathcal{C}}(D_Q))$ can be encoded into a positive logic program. As a set of facts, the program contains the atoms in D_Q. Then we extend our signature by two additional relation symbols R^i and R^a for every R in Σ, to be able to reason about the ideal and available instances. We also introduce a *copy rule* r_R: $R^i(\bar{X}) \leftarrow R(\bar{X})$ for every relation and denote the set of all such copy rules as P_Σ. Thus, any model of D_Q and P_Σ contains an "ideal" copy of D_Q.

Next, to capture the reasoning with TC statements, we introduce for each $C \in \mathcal{C}$, $C = \mathit{Compl}(R(\bar{s}); G)$, the rule r_C, defined as:

$$R^a(\bar{s}) \leftarrow R^i(\bar{s}), G^i,$$

where the i in G^i indicates that all relation symbols are replaced by their ideal version. For example, $r_{C_{\text{sci}}}$ is the rule $\mathit{pupil}^a(N, C, S) \leftarrow \mathit{pupil}^i(N, C, S), \mathit{class}^i(C, S, L, \mathit{sci})$ and $r_{C_{\text{lev1}}}$ the rule $\mathit{class}^a(C, S, 1, B) \leftarrow \mathit{class}^i(C, S, 1, B)$. The set of all rules for statements in $\mathcal{C}$ is

$$P_{\mathcal{C}} = \{r_C \mid C \in \mathcal{C}\}.$$

Intuitively, D_Q is a prototypical instance where Q returns an answer, namely, the prototypical answer $\theta\bar{X}$. Applying the copy rules in P_Σ turns D_Q into an ideal database D^i_Q. In the following, we use $\mathit{ideal}(D)$ or D^i (resp., $\mathit{avail}(D)$ or D^a) to denote the ideal copy (resp., the available copy) of the instance D. The application of the statement rules in $P_{\mathcal{C}}$ then amounts to computing the available instance $\mathit{avail}(T_{\mathcal{C}}(D^i_Q))$. It remains to check whether Q returns the answer $\theta\bar{X}$ also over $\mathit{avail}(T_{\mathcal{C}}(D^i_Q))$.

To this end, we introduce the boolean *test query* Q^s, which is obtained from Q by replacing each relation symbol by its available version and freezing the *distinguished* variables. Formally, Q^s is defined by the rule $r_{Q^s} = Q^s \leftarrow R^a_1(\theta\bar{t}_1), \ldots, R^a_n(\theta\bar{t}_n)$, where θ is the substitution that maps every *distinguished* variable to its frozen version. In our example, the test query is $Q^s_{\text{s1n}} \leftarrow \mathit{pupil}^a(n', C, \mathit{newton}), \mathit{class}^a(C, \mathit{newton}, 1, \mathit{sci})$.

In addition to the program $P_{\mathcal{C}}$, encoding $\mathcal{C}$, we define $P^s_Q := D_Q \cup P_\Sigma \cup \{r_{Q^s}\}$, as the program encoding Q.

THEOREM 3 (SET ENCODING). *Let Q be a conjunctive query and $\mathcal{C}$ be a set of TC statements. Then $\mathcal{C} \models \mathit{Compl}^s(Q)$ iff*

1. *the atom Q^s is in the answer set of $P_{\mathcal{C}} \cup P^s_Q$, or*
2. *the goal Q^s succeeds with $P_{\mathcal{C}} \cup P^s_Q$ under SLD resolution.*

PROOF. 1. To prove the claim it is sufficient to show that $\theta\bar{X} \in Q(T_{\mathcal{C}}(D_Q))$ if and only if the atom Q^s is in the answer set of $P_{\mathcal{C}} \cup P^s_Q$. From this and Theorem 2 the claim follows directly. Since $P_{\mathcal{C}} \cup P^s_Q$ is a positive program, it has a unique answer set that corresponds to the least fixed point (LFP) of the program. By the program definition, the LFP of the program is consists of the facts in

(i) D_Q, (ii) the ideal copy D^i_Q produced by $\mathcal{P}_\Sigma$, (iii) the available version $avail(T_\mathcal{C}(D_Q))$ of $T_\mathcal{C}(D_Q)$, obtained by applying the program $\mathcal{P}_\mathcal{C}$ on D^i_Q, and (iv) possibly the ground atom Q_s that can be produced by the rule r_{Q^s} over the facts from $avail(T_\mathcal{C}(D_Q))$. Thus, we have that Q^s is in the LFP of the encoding program if and only if $\theta\bar{X} \in Q(T_\mathcal{C}(D_Q))$.

2. For SLD resolution it holds that each SLD-provable ground atom of a program belongs to the least Herbrand model (LHM) of the program. Since, in our case, the LFP of the answer set program is the same as the LHM of the Prolog program, the claim follows from Case 1. □

Recall that in the definition of the test query, we *freeze* the distinguished variables because we want to test whether Q returns $\theta\bar{X}$, and we do *not freeze* the non-distinguished variables because we do not want to impose constraints on *how* $\theta\bar{X}$ is retrieved. To see why this definition works, consider the query

$$Q_{\text{cht}}(N) \leftarrow takes(N, chess), takes(N, A), \tag{5}$$

which asks for "all pupils that take chess, which in addition take some activity" (which may be chess). Suppose, our data are complete for all pupils that take chess, expressed by the TC statement C_{ch}, that has rule form $takes^a(N, chess) \leftarrow takes^i(N, chess)$. The ideal copy of the canonical database $D_{Q_{\text{cht}}}$ is $D^i_{Q_{\text{cht}}} = \{takes^i(n', chess),\ takes^i(n', a')\}$. With our TC rule, we can derive the fact $takes^a(n', chess)$, but not $takes^a(n', a')$. Still, this is enough to succeed for our test query $Q^s_{\text{cht}} \leftarrow takes^a(n', chess), takes^a(c', A)$, since the variable A can be bound to *chess*. Thus, the check in Theorem 3 returns the intuitively expected answer "yes".

Completeness Under Bag Semantics. Under bag semantics, a query returns an answer tuple as many times as there are satisfying assignments for that tuple. Thus, a set $\mathcal{C}$ may entail completeness of Q under set, but not under bag semantics. The query Q_{cht} in (5) returns each pupil that takes chess as many times as there are activities taken by that pupil. Thus, under bag semantics the additional atom $takes(N, A)$ makes sense, as it does not change the answers as such, but the multiplicity with which they appear. The query now asks, "How many activities does each chess taker take?"

For reasoning under bag semantics, it no longer suffices to check whether each answer of Q over D^i is also an answer over D^a, but whether each *satisfying assignment* for Q over D^i is retained over D^a. Intuitively, the freezing assignment θ is a prototypical satisfying assignment for Q over the prototypical ideal database D_Q. The minimal available database that, together with D_Q, satisfies $\mathcal{C}$, is $T_\mathcal{C}(D_Q)$. To test whether θ is also a satisfying assignment over $T_\mathcal{C}(D_Q)$, we simply check whether $D_Q \subseteq T_\mathcal{C}(D_Q)$.

THEOREM 4 (CHARACTERIZATION BAG SEMANTICS). *Let $\mathcal{C}$ be a set of TC statements, and Q be a conjunctive query. Then*

$$\mathcal{C} \models Compl^b(Q) \iff D_Q \subseteq T_\mathcal{C}(D_Q).$$

PROOF. ($\Rightarrow$) As in the proof of Theorem 2, we observe that $(D_Q, T_\mathcal{C}(D_Q)) \models \mathcal{C}$. Hence, $(D_Q, T_\mathcal{C}(D_Q)) \models Compl^b(Q)$, that is, the number of times a tuple occurs as an answer of Q over $T_\mathcal{C}(D_Q)$ is the same it occurs over D_Q. Since $T_\mathcal{C}(D_Q) \subseteq D_Q$, this is only possible if every assignment that satisfies Q over D_Q also satisfies Q over $T_\mathcal{C}(D_Q)$. In particular, the freezing mapping θ is an assignment that satisfies Q over $D_Q = \theta B$ where B is the body of Q. By the argument above, θ also satisfies Q over $T_\mathcal{C}(D_Q)$, and hence $\theta B \subseteq T_\mathcal{C}(D_Q)$, that is, $D_Q \subseteq T_\mathcal{C}(D_Q)$.

($\Leftarrow$) Suppose $D_Q \subseteq T_\mathcal{C}(D_Q)$. Let $\mathcal{D} = (D^i, D^a) \models \mathcal{C}$ for an incomplete database $\mathcal{D}$ and let $\bar{c} \in Q(D^i)$. We show that $\bar{c}$ appears the same number of times as answer to Q over D^a as it does over D^i. Since $D^a \subseteq D^i$, it suffices to show that each assignment α returning $\bar{c}$ over D^i also returns $\bar{c}$ over D^a. Similar to the proof for the case of set semantics, one can show that $T_\mathcal{C}(\alpha B) \subseteq D^a$. Further, from $D_Q \subseteq T_\mathcal{C}(D_Q)$ it follows that $\alpha\theta^{-1}(\theta B) \subseteq \alpha\theta^{-1}T_\mathcal{C}(\theta B)$. Next, $\alpha\theta^{-1}T_\mathcal{C}(\theta B) \subseteq T_\mathcal{C}(\alpha\theta^{-1}\theta B) = T_\mathcal{C}(\alpha B)$ for the same reasons as in the proof of Theorem 2. Altogether, $\alpha B \subseteq T_\mathcal{C}(\alpha B)$. Then, from $\alpha B \subseteq T_\mathcal{C}(\alpha B) \subseteq D^a$ we conclude $\alpha B \subseteq D^a$. This implies that α is also a satisfying assignment for Q over D^a, which returns $\bar{c}$. □

This characterization can be readily encoded into a logic program. Instead of freezing only the *distinguished* variables of Q, we freeze *all* variables in Q. We thus define Q^b by the rule $r_{Q^b} = Q^b \leftarrow R^a_1(\theta\bar{t}_1), \ldots, R^a_n(\theta\bar{t}_n)$, where now θ is the substitution that maps *every* variable to its frozen version. The encoding of Q is now $P^b_Q := D_Q \cup P_\Sigma \cup \{r_{Q^b}\}$.

THEOREM 5 (BAG ENCODING). *Let Q be a conjunctive query and $\mathcal{C}$ be a set of TC statements. Then $\mathcal{C} \models Compl^b(Q)$ iff*

1. *the atom Q^b is in the answer set of $P_\mathcal{C} \cup P^b_Q$, or*
2. *the goal Q^b succeeds with $P_\mathcal{C} \cup P^b_Q$ under SLD resolution.*

PROOF SKETCH. 1. Due to Theorem 4, it suffices to prove that the atom Q^b is in the answer set of $P_\mathcal{C} \cup P^b_Q$ iff $D_Q \subseteq T_\mathcal{C}(D_Q)$. Similarly to the case of set semantics, this can be proved by showing that the test $D_Q \subseteq T_\mathcal{C}(D_Q)$ succeeds if and only if the rule r_{Q^b} fires.

2. As for Theorem 3, the claim follows from the fact that a ground atom can be SLD-resolved iff it belongs to the least Herbrand model (LHM) of the program. In our encoding, the LHM corresponds to least fixed point of the program. Thus, the claim follows from Claim 1. □

The tests in Theorems 2 and 4 can be performed in nondeterministic polynomial time, which complies with the overall complexity of the problems, which are NP-complete.

5. REASONING UNDER FOREIGN KEYS

Foreign keys can help us draw additional conclusions about query completeness. We provide an example for both the effects of unenforced and enforced FKs.

Example. Consider the query $Q_{\text{n}}(N) \leftarrow pupil(N, C, newton)$. Recall that for each class, identified by its code and the name of its school, the relation *class* records the level and branch of that class. Assume that our database is "complete for those pupils at Newton school that attend a class that belongs to some level and branch," expressed as $Compl(pupil(N, C, newton); class(C, newton, L, B))$. If there may exist classes at Newton school present in the *pupil* relation but not in *class* relation, then the completeness of Q_{n} does not follow. However, if we know that every class in the application domain has a level and belongs to a branch, we can express this by stating that the FK $pupil[ccode, sname] \subseteq class[ccode, sname]$ holds (without being enforced). In this case, for every pupil record there exists a corresponding class record in the ideal database, hence the above TC statement is enough to guarantee the completeness of Q_{n}.

However, if we ask for all pupils attending a class of the science branch at Newton school, $Q_{\text{sn}}(N) \leftarrow pupil(N, C, newton), class(C, newton, L, sci)$, then the non-enforced FK does not guarantee completeness, since then for every *pupil* record a corresponding *class* record exists in the ideal database, representing the state of the real world, but such *class* records may not be present in the available database so that we cannot identify classes of the science branch. If

the FK holds *and* is enforced, then all such records must be present, and the completeness of Q_{sn} is ensured.

We will characterize query completeness in the presence of key and weakly acyclic FK constraints, both for non-enforced and enforced semantics. We assume that the bodies of the conjunctive queries to be checked for completeness satisfy the key constraints, that is, if A is the list of key attributes of R and if $R(\bar{s})$ and $R(\bar{t})$ are two atoms in R such that $R(\bar{s})[A] = R(\bar{t})[A]$, then $\bar{s} = \bar{t}$. Satisfaction of key constraints can be achieved efficiently by chasing the body with the key constraints (see [12]).

The conceptual tool needed for foreign key reasoning is the chase with inclusion dependencies (IDs), which generalize foreign-key constraints. Johnson and Klug defined two versions, the *oblivious* chase, which introduces a new atom whenever an ID is "applicable," and the *restricted* or *standard* chase, which only introduces a new atom for an applicable ID if the ID is not yet satisfied [12]. We will use an intermediate chase version, introduced by Marnette [16], the *oblivious Skolem chase*, which generates atoms with Skolem terms whenever an ID is applicable, and the atom to be generated does not exist yet. The oblivious Skolem chase bears some similarity with the restricted chase in that a chase step is blocked if the corresponding ID is satisfied by an atom with Skolem terms.

To apply this chase, IDs are translated into rules with Skolem terms in the head. For instance, the foreign key from pupil to class is translated into the rule

$$\mathit{class}(C,S,f_{\mathit{class,level}}(C,S),f_{\mathit{class,branch}}(C,S)) \leftarrow \mathit{pupil}(N,C,S), \quad (6)$$

which introduces a new class atom for every pupil atom, with a new level and branch generated by two Skolem functions that take the keys of the class atom as arguments. Let $\mathcal{K}$ be a set of FKs. We denote the result of chasing a set of (ground) atoms D with the rules obtained from $\mathcal{K}$ as $Ch_{\mathcal{K}}(D)$. If $\mathcal{K}$ is clear from the context, we drop the subscript and write $Ch(D)$. In general, chase termination is not guaranteed for FKs.

From now on, we consider only weakly acyclic sets of FKs [7]. Weakly acyclic tuple generating dependencies (which are more general constraints than FKs) have been introduced in the context of data exchange as syntactic class that ensures termination of the restricted chase for every instance [7]. Marnette extended this result to the oblivious Skolem chase [16]. Under this assumption, $Ch_{\mathcal{K}}(D)$ is always finite for a finite D.

Now, we can formulate a result for non-enforced FKs that generalizes Theorems 2 and 4 by replacing the prototypical database D_Q with the chase of D_Q.

THEOREM 6 (CHARACTERIZATION NON-ENFORCED FKS). *Let $\mathcal{C}$ be a set of TC statements, $\mathcal{K}$ be a weakly acyclic set of FKs, and Q be a conjunctive query. Then*

1. $\mathcal{C} \models_{\mathcal{K}} \mathit{Compl}^s(Q) \iff \theta\bar{X} \in Q(T_{\mathcal{C}}(Ch_{\mathcal{K}}(D_Q)))$
2. $\mathcal{C} \models_{\mathcal{K}} \mathit{Compl}^b(Q) \iff D_Q \subseteq T_{\mathcal{C}}(Ch_{\mathcal{K}}(D_Q))$.

PROOF IDEA. 1. ($\Rightarrow$) The proof is based on a similar construction as the one for Theorem 2. In this case we take $\mathcal{D}_Q = (Ch(D_Q), T_{\mathcal{C}}(Ch(D_Q)))$ as an incomplete database.

($\Leftarrow$) This direction is more involved. We have to cope with the difficulty that applying our chase to a database instance that satisfies the key constraints in $\mathcal{K}$ may result in an instance that no more satisfies the keys because of additional atoms with Skolem terms. For instance, chasing $\{\mathit{pupil}(\mathit{fred},a1,\mathit{newton}), \mathit{class}(a1,\mathit{newton},1,\mathit{sci})\}$ with rule (6) adds the atom $\mathit{class}(a1,\mathit{newton},f_{\mathit{class,level}}(a1,\mathit{newton}),f_{\mathit{class,branch}}(a1,\mathit{newton}))$, so that the key constraint for *class* is violated.

We compensate this proliferation of atoms by introducing for $\mathcal{K}$ a new set inclusion relation $\subseteq_{\mathcal{K}}$ (and based on that an equivalence relation $\equiv_{\mathcal{K}}$) that regards atoms as identical if the keys implies they are the same. With respect to this inclusion relation, the operator $Ch_{\mathcal{K}}()$ is idempotent and monotonic. Moreover, for any constant tuple $\bar{c}$, query Q, and database instances D, D', possibly with Skolem terms, we have that $\bar{c} \in Q(D)$ implies $\bar{c} \in Q(D)$, whenever $D \subseteq_{\mathcal{K}} D'$. Then the proof of Theorem 2 can be generalized to the new setting, replacing set inclusion and equality with $\subseteq_{\mathcal{K}}$ and $\equiv_{\mathcal{K}}$.

2. Can be shown in the similar way as Claim 1 by generalizing the ideas in the proof of Theorem 2. □

If the FKs are enforced, we also need to extend the available database obtained by applying $T_{\mathcal{C}}$ to the chase of D_Q to satisfy $\mathcal{K}$. However, this cannot be done by chasing this result a second time. To see this, consider the query Q_{sn} from above and the TC statement $C_p = \mathit{Compl}(\mathit{pupil}(N,C,S); \mathit{true})$, which states that the table *pupil* is complete. Together with the enforced FK from *pupil* to *class* this implies completeness of Q_{sn}, since then any available database must contain (i) all pupils and (ii) the class of each pupil. Let us try to reach this consequence by reasoning about the prototypical database for Q_{sn}. Clearly, $D_{Q_{\text{sn}}} = \{\mathit{pupil}(n,c,\mathit{newton}), \mathit{class}(c,\mathit{newton},l,\mathit{sci})\}$ and $Ch_{\mathcal{K}}(D_{Q_{\text{sn}}}) = D_{Q_{\text{sn}}}$, since there is already a class for the pupil in $D_{Q_{\text{sn}}}$. Applying $T_{\mathcal{C}}$ yields $T_{\mathcal{C}}(Ch_{\mathcal{K}}(D_{Q_{\text{sn}}})) = \{\mathit{pupil}(n,c,\mathit{newton})\}$, since we are complete for *pupil*, but not for *class*. The chase would add a new *class* atom, say, $\mathit{class}(c,\mathit{newton},l',b')$. However, Q_{sn} will not retrieve the pupil n over the database $\{\mathit{pupil}(n,c,\mathit{newton}), \mathit{class}(c,\mathit{newton},l',b')\}$, because it looks for pupils in a science class.

Clearly, instead of generating a new *class* atom, we should have copied the atom $\mathit{class}(c,\mathit{newton},l,\mathit{sci})$ from the ideal db to satisfy the FK. To achieve this, we define the chase operator $Ch^a_{\mathcal{K}}(\cdot)$, which, given an incomplete database (D^i,D^a) where D^i satisfies $\mathcal{K}$, extends D^a to satisfy $\mathcal{K}$ by copying atoms from D^i. Formally, if $R(\bar{s}) \in D^a$ and if $R[A] \subseteq S[B] \in \mathcal{K}$, we add *all* atoms $S(\bar{t}) \in D^i$ to D^a where $S(\bar{t})[B] = R(\bar{s})[A]$. The result is denoted as $Ch^a_{\mathcal{K}}(D^i,D^a)$.

To formalize the reasoning performed in the previous example, we combine the two chase operators and the $T_{\mathcal{C}}$ operator to a new operator $\mathcal{K}_{\mathcal{C}}$, defined as $\mathcal{K}_{\mathcal{C}}(D) = Ch^a_{\mathcal{K}}(Ch(D), T_{\mathcal{C}}(Ch(D)))$. This means that (i) we chase D to obtain an ideal db that satisfies the foreign keys in $\mathcal{K}$, (ii) apply $T_{\mathcal{C}}$, and (iii) extend the result by copying atoms from $Ch_{\mathcal{K}}(D)$.

THEOREM 7 (CHARACTERIZATION ENFORCED FKS). *Let $\mathcal{C}$ be a set of TC statements, $\mathcal{K}$ be a weakly acyclic set of FKs, and Q be a conjunctive query. Then*

1. $\mathcal{C} \models^e_{\mathcal{K}} \mathit{Compl}^s(Q) \iff \theta\bar{X} \in Q(\mathcal{K}_{\mathcal{C}}(D_Q))$
2. $\mathcal{C} \models^e_{\mathcal{K}} \mathit{Compl}^b(Q) \iff D_Q \subseteq \mathcal{K}_{\mathcal{C}}(D_Q)$.

The proof generalizes the techniques for showing Theorem 6. Now we extend our encoding so that it supports reasoning under FKs. Since a partial database satisfies an FK constraint if the ideal database satisfies it, we introduce for every FK constraint a rule like the one in (6), albeit for the ideal version of the relations involved. For instance, instead of (6), we introduce the rule $\mathit{class}^i(C,S,f_{\mathit{class,level}}(C,S),f_{\mathit{class,branch}}(C,S)) \leftarrow \mathit{pupil}^i(N,C,S)$. We denote the set of all FK rules obtained from $\mathcal{K}$ as $P^i_{\mathcal{K}}$, where $\cdot^i$ indicates that the rules create atoms for the ideal instance.

Finally, to model reasoning about enforced foreign keys, we introduce rules that copy foreign key atoms from the ideal to the available database. For example, the rule for the FK from *pupil* to *class* is

$$\mathit{class}^a(C,S,L,B) \leftarrow \mathit{pupil}^a(N,C,S), \mathit{class}^i(C,S,L,B).$$

We denote the set of all such rules obtained from $\mathcal{K}$ as $P^e_{\mathcal{K}}$, where $\cdot^e$ indicates enforcement.

We are now ready to state the encoding theorems for reasoning about FKs, whose proofs are consequences of the Characterization Theorems 6 and 7.

THEOREM 8 (SET ENCODING FOR NON-ENFORCED FKS). *Let Q and $\mathcal{C}$ be as previously. Let $\mathcal{K}$ be a weakly acyclic set of FK constraints. Then $\mathcal{C} \models_{\mathcal{K}} Compl^s(Q)$ iff*

1. *the atom Q^s is in the answer set of $P_{\mathcal{C}} \cup P^i_{\mathcal{K}} \cup P^s_Q$, or*
2. *Q^s succeeds with $P_{\mathcal{C}} \cup P^i_{\mathcal{K}} \cup P^s_Q$ under SLD resolution.*

Similar Encoding Theorems hold for any combination of set/bag semantics and non-enforced/enforced semantics. For bag semantics, we have to replace the rule set P^s_Q by P^b_Q and for enforced semantics we have to add the rule set $P^e_{\mathcal{K}}$.

6. REASONING UNDER CONDITIONAL FINITE DOMAIN CONSTRAINTS

Finite domain constraints make it necessary to reason by cases.

Example. We take up again the example from the introduction, with the query

$$Q_{\text{sn}}(N) \leftarrow pupil(N,C,newton), class(C,newton,L,sci),$$

asking for the pupils from the science branch at Newton school, the CFDC $F_{\text{sn}} = class\{school = newton, branch = sci\}[ccode] = \{a1,b2\}$, stating that the only science classes at Newton school are *a1* and *b2*, and the TC statements $C_{\text{a1n}} = Compl(pupil(N, a1, newton); true)$ and $C_{\text{b2n}} = Compl(pupil(N,b2,newton); true)$, stating completeness for the pupils of classes *a1* and *b2*. The TC statements alone do not entail completeness of Q_{sn}, because in principle there could be other science classes. However, together with the CFDC, they do imply completeness.

The characterization in Theorem 2, though, does not apply here because the operators $T_{C_{\text{a1n}}}$ and $T_{C_{\text{b2n}}}$ map the frozen body of Q_{sn} to the empty set. They transfer *pupil* atoms where the class code is *a1* or *b2*. Therefore, a reasoning procedure has to instantiate the codes in all ways permitted by the CFDC before applying the $T_{\mathcal{C}}$ operator. If the test of Theorem 2 succeeds for all such instantiations, then we can conclude completeness.

In the following, we develop concepts that allow us to characterize completeness wrt CFDCs and then provide an encoding of the characterizing condition. Since CFDCs are essentially disjunctive, we encode reasoning only into answer set programming, which supports disjunctive rules, while Prolog does not.

Let $\mathcal{F}$ be a fixed set of CFDCs. Suppose B is a conjunction of atoms. Let γ be a substitution that maps each variable Y of B either to a constant occurring in $\mathcal{F}$ (a *proper* constant) or to the frozen version of Y. We say that γ *satisfies B and $\mathcal{F}$* if the instance γB satisfies $\mathcal{F}$. We say that a substitution γ' is *at least as general* as γ, written $\gamma' \succ \gamma$, if for all variables Y in B and all proper constants c we have that $\gamma' Y = c$ implies $\gamma Y = c$. We say that γ is an *$\mathcal{F}$-case of C* if γ is maximally general among all substitutions satisfying B and $\mathcal{F}$. For instance, the body of the query Q_{sn} has two $\{F_{\text{sn}}\}$-cases, γ_{a1}, which substitutes the variable C with *a1* and freezes all other variables, and γ_{b2}, which is analogously defined. By $\Gamma_{\mathcal{F},B}$ we denote the set of all $\mathcal{F}$-cases of B.

We remark that by a reduction of the 3-colorability problem one can construct a set $\mathcal{F}_{\text{3col}}$ of CFDCs such that it is NP-hard to decide whether $\Gamma_{\mathcal{F}_{\text{3col}},B} \neq \emptyset$ for a conjunction B. It is also not difficult to see that in polynomial time one can verify whether a substitution is an $\mathcal{F}$-case of B. Thus, checking whether $\Gamma_{\mathcal{F},B} \neq \emptyset$ is NP-complete. We now generalize Theorem 2 to CFD constraints. We drop the analogous generalization of Theorem 4 due to the limited space.

THEOREM 9 (CHARACTERIZATION CFD CONSTRAINTS). *Let $\mathcal{C}$ be a set of TC statements, $\mathcal{F}$ be a set of CFD constraints, and $Q(\bar{X}) \leftarrow B$ be a conjunctive query. Then*

$$\mathcal{C} \models_{\mathcal{F}} Compl^s(Q) \iff \gamma\bar{X} \in Q(T_{\mathcal{C}}(\gamma B)) \text{ for every case } \gamma \text{ in } \Gamma_{\mathcal{F},B}. \quad (7)$$

PROOF IDEA. For $\gamma \in \Gamma_{\mathcal{F},B}$ we define the substitution $\tilde{\gamma}$ as $\tilde{\gamma}Y = \gamma Y$ if γY is a constant, and $\tilde{\gamma}Y = Y$ otherwise. That is, $\tilde{\gamma}$ substitutes variables that are subject to a CFD constraint by one of the possible values and leaves the other variables as they are. The proof then exploits the fact that over any database satisfying $\mathcal{F}$, a query $Q(\bar{X}) \leftarrow B$ is equivalent to the union of conjunctive queries $\bigcup_{\gamma \in \Gamma_{\mathcal{F},B}} Q_\gamma$, where each Q_γ is defined as $Q_\gamma(\tilde{\gamma}\bar{X}) \leftarrow \tilde{\gamma}B$. □

Condition (7) stipulates a test comprising the following steps: (i) Instantiate the query body B in all possible ways permitted by the CFD constraints. This amounts to considering all possible ways in which an answer to Q can be retrieved. (ii) For each such instantiation γB, compute $T_{\mathcal{C}}(\gamma B)$, the set of atoms that must be present in any available database if the ideal database contains γB. (iii) Evaluate Q over $T_{\mathcal{C}}(\gamma B)$ and check whether the result contains $\gamma\bar{X}$, the prototypical answer to Q over γB. We now show how to encode (i) $\mathcal{F}$-cases, (ii) the operator $T_{\mathcal{C}}$, and (iii) the final test into disjunctive rules.

The starting point is again the canonical database D_Q. Since D_Q consists of ground atoms, which cannot be instantiated, we have to mimic the instantiation by cases. To this end, we introduce a new binary predicate *val*, where intuitively $val(t,v)$ indicates that the term t is instantiated by the value v. To achieve modularity, we keep the encodings of Q, $\mathcal{F}$, and $\mathcal{C}$ independent from each other. Therefore, no information as to which variable can be instantiated by which value should influence the encoding of Q. We achieve this by introducing $val(t,t)$ for each term in D_Q both for original constants and for frozen variables. The set of all such facts is denoted as Val_Q. Then we require that every term can have at most one *val*-value other than itself, which can be seen as a requirement for *val* to be functional. This can be expressed by the denial r_{fun}:

$$\leftarrow val(X,Y), val(X,Z), X \neq Y, X \neq Z, Y \neq Z.$$

To mimic the instantiation of the query body by cases, we introduce for each CFD constraint $F = R\{a_1 = v_1, \ldots, a_n = v_n\}[a] = \{w_1, \ldots, w_m\}$ the disjunctive rule r_F as

$$val(X_a,w_1) \mid \cdots \mid val(X_a,w_m) \leftarrow R^i(X_1,\ldots,X_a,\ldots X_n), val(X_{a_1},v_1),\ldots,val(X_{a_n},v_n),$$

which nondeterministically binds the term in attribute position a to one of the values in $\{w_1,\ldots,w_m\}$, provided the attribute positions $a_1,\ldots,a_n$ are bound to $v_1,\ldots,v_n$, respectively. Thus, the CFD constraint F_{sn} is translated into the rule $val(C,a1) \mid val(C, b2) \leftarrow class(C,S,L,B), val(S,newton), val(B,sci)$. We collect the rules encoding $\mathcal{F}$ into the program $P_{\mathcal{F}} := \{r_F \mid F \in \mathcal{F}\} \cup \{r_{\text{fun}}\}$.

We keep the rules P_Σ that copy D_Q into the ideal database D^i_Q. However, to make the TC rules applicable to facts with *val*-bindings, we need to unfold them. For an atom $A = R(t_1,\ldots,t_n)$, the *unfolding* consists of the atom $A^u = R(Y_1,\ldots,Y_n)$, where each argument is replaced by a fresh variable, and the set of *val*-bindings $U^A = \{val(Y_1,t_1),\ldots,val(Y_n,t_n)\}$. Earlier, we have shown how

to translate a TC statement C into the rule r_C: $R^a(t_1,\ldots,t_n) \leftarrow R^i(t_1,\ldots,t_n), G^i$. The *unfolded rule* for C is then

$$r^u_C: \quad R^a(Y_1,\ldots,Y_n) \leftarrow R^i(Y_1,\ldots,Y_n), U^{R^i(t_1,\ldots,t_n)}, (G^i)^u, U^{G^i},$$

where the unfolding $(G^i)^u$ of the condition G^i and the set of value atoms U^{G^i} are defined analogously to the case of single atoms. For example the rule $r^u_{C_{\text{sci}}}$ for C_{sci} in (2) is

$$pupil^a(N_1,C_1,S_1) \leftarrow$$
$$pupil^i(N_1,C_1,S_1), val(N_1,N), val(C_1,C), val(S_1,S),$$
$$class^i(C_2,S_2,L_2,B_2), val(C_2,C), val(S_2,S), val(L_2,L), val(B_2,sci).$$

The program consisting of the unfolded rules is denoted as $P^u_{\mathcal{C}} = \{r^u_C \mid C \in \mathcal{C}\}$.

Let $Q(X_1,\ldots,X_n) \leftarrow B$ be the query which we want to check for completeness. We have to modify the test query in two ways, first by unfolding, and second by adding *val*-atoms to encode the check whether Q returns $\gamma\bar{X}$ over $T_{\mathcal{C}}(\gamma B)$. Thus, the rule $r_{Q^u_s}$ for the test query Q^u_s is

$$Q^u_s \leftarrow (B^a)^u, U^{B^a}, val(x_1,X_1),\ldots,val(x_n,X_n),$$

where the x_j are the frozen versions of the X_j. For example, the test query for Q_{sn} in (7) is

$$Q^u_{\text{sn}} \leftarrow pupil^a(N_1,C_1,S_1), class^a(C_2,S_2,L_2,B_2), val(N_1,N), val(n',N), val(C_1,C), val(S_1,newton), val(C_2,C), val(S_2,newton), val(L_2,L), val(B_2,sci),$$

where n' is the frozen version of N. Let $P^{s,u}_Q := D^i_Q \cup Val_Q \cup \{r_{Q^u_s}\}$ be the set of rules about Q.

THEOREM 10 (ASP ENCODING OF CFDCS). *Let Q be a conjunctive query, $\mathcal{C}$ a set of TC statements, and $\mathcal{F}$ a set of CFD constraints. Then*

$$\mathcal{C} \models_{\mathcal{F}} Compl^s(Q) \iff Q^u_s \text{ is in every answer set of } P^u_{\mathcal{C}} \cup P_{\mathcal{F}} \cup P^{s,u}_Q.$$

As in Theorem 5, we obtain an analogous encoding of bag completeness checks by modifying the rule for the test query. In order to ensure that no assignment is lost, we add to the body of the rule for Q^u_b for every variable Y in Q an atom $val(y,Y)$ with the frozen version y of Y.

7. FOREIGN KEYS AND CFDCS

Foreign keys, in particular combined with conditional finite domains, allow for a larger set of inferences.

Example. Consider $Q_{\text{n}}(N) \leftarrow pupil(N,C,newton)$, which asks for the names of all pupils at Newton school, and assume the FK and CFD constraints from our running example hold. Consider as well the TC statement C_{sci} in (2) and an analogous statement C_{hum}, which asserts completeness for pupils in humanities classes. Now, Q_{n} is complete because every pupil of Newton school belongs to a class of some branch (by the FKs), the only possible branches are *sci* and *hum* (by the CFDCs), and we are complete for each by C_{sci} and C_{hum}. Note that we arrived at this conclusion without assuming that the FKs are enforced.

We now sketch how the constructions in Sections 5 and 6 can be combined to reasoning about CFDCs and FKs in conjunction. First we provide a characterization of completeness under set semantics. The difference with respect to Theorem 9 is now that we have to consider all cases of the chase of the body B of Q instead of B alone. One can show that it still does not matter which version of the chase is used.

THEOREM 11 (CHARACTERIZATION). *Let $\mathcal{C}$ be a set of TC statements, $\mathcal{K}$ be a weakly acyclic set of FKs, $\mathcal{F}$ be a set of CFDCs, and $Q(\bar{X}) \leftarrow B$ be a conjunctive query. Then*

$$\mathcal{C} \models_{\mathcal{K},\mathcal{F}} Compl^s(Q) \iff \gamma\bar{X} \in Q(T_{\mathcal{C}}(\gamma(Ch_{\mathcal{K}}(B)))) \text{ for every case } \gamma \text{ in } \Gamma_{\mathcal{F},Ch_{\mathcal{K}}(B)}.$$

The characterization theorem for enforced FKs is a modification of Theorem 7 where Ch^a is to be applied to $(\gamma(Ch_{\mathcal{K}}(B)), T_{\mathcal{C}}(\gamma(Ch_{\mathcal{K}}(B))))$ for every γ in $\Gamma_{\mathcal{F},Ch_{\mathcal{K}}(B)}$.

For the encoding, there are now two modifications with respect to the case of plain FKs. The first is that every new term that is created by a chase rule must be a value of itself. The second is that the copy rules for enforced semantics need to be unfolded. For example, for the FK from *pupil* to *class*, we have to add the rule

$$val(f_{class,level}(C,S), f_{class,level}(C,S)) \leftarrow pupil^i(N,C,S),$$

which makes every new level term a value of itself. A similar rule is also needed for the branch of a class.

Let $\mathcal{K}$ be a set of FKs. We denote the extension of $P_{\mathcal{K}}$ by these rules as $P^u_{\mathcal{K}}$. Similarly, we denote the set of unfolded versions of the copy rules as $P^{e,u}_{\mathcal{K}}$. With these rules we can now encode completeness reasoning that considers both FKs and CFDCs into answer set programming.

THEOREM 12 (ASP ENCODING). *Let Q, $\mathcal{C}$, and $\mathcal{F}$ as before and let $\mathcal{K}$ be a weakly acyclic set FK constraints. Then*

1. $\mathcal{C} \models_{\mathcal{F},\mathcal{K}} Compl^s(Q) \iff Q^u_s$ *is in every answer set of* $P^u_{\mathcal{C}} \cup P_{\mathcal{F}} \cup P^{i,u}_{\mathcal{K}} \cup P^{s,u}_Q$;
2. $\mathcal{C} \models^e_{\mathcal{F},\mathcal{K}} Compl^s(Q) \iff Q^u_s$ *is in every answer set of* $P^u_{\mathcal{C}} \cup P_{\mathcal{F}} \cup P^{i,u}_{\mathcal{K}} \cup P^{e,u}_{\mathcal{K}} \cup P^{s,u}_Q$.

An analogous result holds for bag semantics.

8. EXPERIMENTS

We performed experiments to find out what is the maximal size of problems that can be solved with our encodings on standard Prolog and Answer Set Programming (ASP) platforms and how the running time depends on the input. Since we expect that in applications there are many constraints that are not relevant to a reasoning problem, we also explored how the performance is influenced by irrelevant information.

We performed four tests and ran them on two state-of-the-art ASP reasoners, `dlv` and `clingo`. We also ran the first test on the `swi` Prolog implementation. The experiments were executed on a desktop machine with an Intel i5 3.40GHz CPU and 8GB memory. Each experiment was repeated ten times and since the time measurements were close, we show here average running times.

For the tests, we extended our schema with two relations: *school* ($\underline{sname}, scluster, type$) and *schoolcluster*($\underline{scluster}, district, status$), which model that every school belongs to some school cluster and is of a certain type, e.g., *school*($newton, cluster5, middle$), and that each school cluster is in some district and has the status public or private, e.g., *schoolcluster*($cluster5, north, pub$).

Test 1. We studied the cost of reasoning with FKs with the query

$$Q_{1\text{pub}}(N) \leftarrow pupil(N,C,S), class(C,S,1,B), \\ school(S,SC,T), schoolcluster(SC,D,pub),$$

which asks for "all 1st level pupils from public clusters." We asserted that we are complete for pupils, expressed by $pupil^a(N,C,S) \leftarrow pupil^i(N,C,S)$, and that we enforced the FKs $pupil[ccode, sname] \subseteq class[ccode, sname]$, $class[sname] \subseteq school[sname]$, and $school[scluster] \subseteq schoolcluster[scluster]$. So, $Q_{1\text{pub}}$ is complete under these assertions.

To simulate reasoning in the presence of irrelevant information, we added TC statements of the form

$$pupil^a(N,\$c,\$s) \leftarrow pupil^i(N,\$c,\$s),$$

where $\$c$ ranged over $\{1,10,10^2\}$ and $\$s$ ranged over $\{1,10,\ldots,10^5\}$, which gave rise to 8 test cases with 10^0 to 10^7 irrelevant TC statements.

The results in Figure 1 show that all systems could deal with sets of up to 10^6 TC statements. While `swi` Prolog could deal also with the largest input, the ASP reasoners timed out after 30 minutes. Moreover, for small and large sets of TC statements, `swi` was one order of magnitude faster than the ASP reasoners.

Test 2. We wanted to assess the cost of reasoning with CFDCs. We ran this test only on ASP reasoners, since implementing CFD reasoning in Prolog requires the usage of meta-programming predicates to cope with disjunctive rules. We reasoned about the query

$$Q_{\text{high}}(N) \leftarrow pupil(N,C,S), class(C,S,L,B), \\ school(S,SC,high),$$

which asks for "all pupils from a class at a high school."

We enumerated all possible schools, class levels, and class branches with FDCS of the form $class\{\}[sname] = \{s_1,\ldots,s_k\}$, $class\{\}[level] = \{l_1,\ldots,l_m\}$, and $class\{\}[branch] = \{b_1,\ldots,b_n\}$. In addition, we asserted that we have complete information about all schools, $Compl(school(S,SC,T); true)$, all school classes, $Compl(class(C,\$S,L,B); true)$, and all pupils in all classes, $Compl(pupil(N,C,\$S); class(C,\$S,\$L,\$B))$, where $\$S$, $\$L$, and $\$B$ range over all school names, class levels, and branches, resp. The combination of the CFD constraints and the TC statements entail that Q_{high} is complete. To conclude completeness, all possible cases combining school name, class level, and branch must be explored. We varied the sizes of these three domains in such a way that the number of cases (and the number of TC statements for class) ranged from 1 to 10^6.

From the results in Figure 2 we observe that for both `clingo` and `dlv` the running times were linear in the size of the input. Both reasoners could cope with up to 1 mio cases. The fastest reasoner, `dlv`, could process 10^4 cases within less than 1 second, which we deem acceptable.

Test 3. We wanted to check whether the running time observed in Test 2 depended on the size of the input or on the number of cases. We constructed an extreme setup where we had a small input, but a large number of cases, and where completeness of the query followed already from few TC statements, while the case analysis was not needed to conclude completeness. We reasoned about the query

$$Q_{\text{pub}}(N) \leftarrow pupil(N,C,S), class(C,S,L,B), \\ school(S,SC,T), schoolcluster(SC,D,pub),$$

which asks for "all pupils from public clusters," in the presence of TC statements that assert the completeness of each of the relations

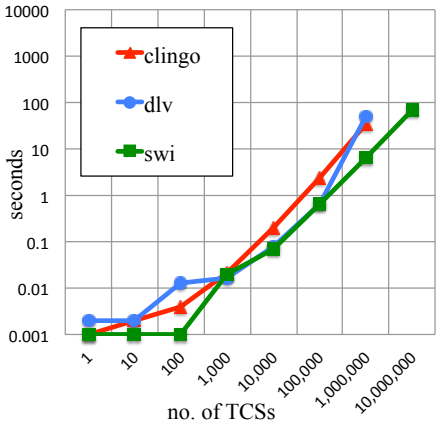

Figure 1: Test 1

in the query, such as $Compl(pupil(N,C,S); true)$, etc. Together, these TCSs guaranteed the completeness of Q_{pub}.

Additionally, we asserted a chain of CFDCs that did not contribute to the completeness of Q_{pub}, but made the reasoning more complicated. Specifically, we stated that there are i different districts, $schoolcluster[district] = \{d_1,\ldots,d_i\}$, that for every such district d there are j different public school clusters, $schoolcluster\{district = d, status = pub\}[scluster] = \{sc_1,\ldots,sc_j\}$, and that for each such cluster there are m different schools, while for each school, there are n different levels.

We varied the parameters i, j, m, n and k to be 1 or 10 (i could be also 100) to obtain different sizes of CFDCs, which gave rise to problems with 10^p answer sets, where $p = 0,\ldots,6$.

Figure 3 displays the running times as depending on the number of answer sets, instead on the input size, which is logarithmically smaller. We observed running times linear in the number of answer sets, which correspond to the number of possible cases, but exponential in the size of the input. The running time, measured in this way, is approximately 10 times faster, than in Test 2, where the case analysis was actually needed. We conclude that reasoning about irrelevant cases takes time, but less than about relevant cases.

Test 4. We wanted to assess the cost of reasoning jointly about FKs and CFDCs. To this end, we slightly modified the setup of Test 3: (i) we dropped the TC statement asserting the completeness of *class*, and (ii) added the enforced FK constraint $pupil[ccode, sname] \subseteq class[ccode, sname]$, which ensures the presence of all class information needed for the query.

The results in Figure 4 show that the number of answer sets that reasoners could cope with before the 30 minutes timeout dropped from 10^6 to 10^4 and that the running time was no more linear in the number of cases.

In summary, our encoding of completeness reasoning allows for efficient reasoning with foreign keys in the presence of large numbers of TC statements, while reasoning about CFDCs depends in the first place on the number of cases covered by a set of CFDCs, rather than the input size. For combinations of FKs and CFDCs, the performance is not yet fully satisfactory.

9. CONCLUSION

We presented techniques to reason about query completeness over partially complete databases satisfying primary and foreign key constraints as well as conditional finite domain constraints. This generalizes previous work on query completeness and allows

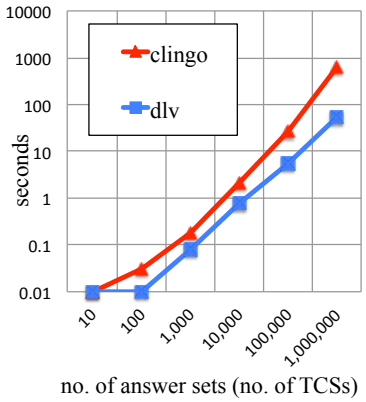

Figure 2: Test 2

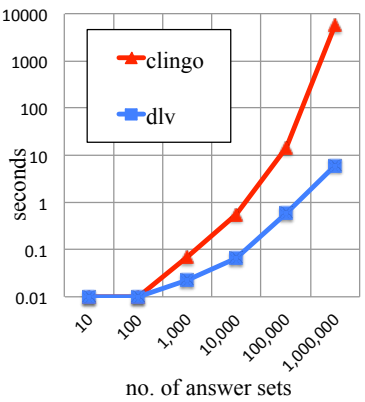

Figure 3: Test 3

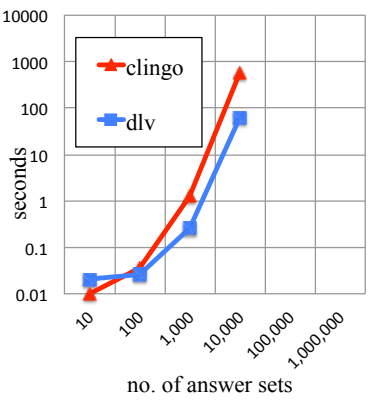

Figure 4: Test 4

one to identify more situations in which queries are complete than was possible before. Our approach also provides a basis for the implementation of these reasoning techniques. We provided syntactic characterizations of completeness reasoning tasks which are independent of any implementation. In a second step, we used them to develop provably correct encodings into logic programming, with and without disjunctive rules, depending on the constraints involved. The encodings can be executed immediately by Prolog engines or ASP solvers.

As a proof of concept, we tested these encodings in a scenario that reflects the requirements of an administrative information system. The results suggest that the encodings are adequate for medium-sized data sets, as they may occur in the administration of a cluster of schools or a university. Currently, we are working on the integration of completeness management into the research information system of a university.

Acknowledgments. This work was partially supported by the project "Managing Completeness of Data (MAGIC)", funded by the province of Bolzano, and the project "Completeness-Aware Querying and Navigation on the Web of Data (CANDy), funded by the Free University of Bozen-Bolzano." We thank the anonymous reviewers for their helpful comments.

10. REFERENCES

[1] S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases*. Addison-Wesley, 1995.

[2] K. Apt and M. van Emden. Contributions to the theory of logic programming. *J. ACM*, 29(3):841–862, 1982.

[3] C. Baral. *Knowledge Representation, Reasoning, and Declarative Problem Solving*. Cambridge University Press, New York, NY, USA, 2003.

[4] A. Chandra and P. Merlin. Optimal implementation of conjunctive queries in relational databases. In *Proc. 9th STOC*, 1977.

[5] A. Cortés-Calabuig, M. Denecker, O. Arieli, and M. Bruynooghe. Approximate query answering in locally closed databases. In *AAAI*, pages 397–402, 2007.

[6] A. Cortés-Calabuig, M. Denecker, O. Arieli, and M. Bruynooghe. Accuracy and efficiency of fixpoint methods for approximate query answering in locally complete databases. In *KR*, pages 81–91, 2008.

[7] R. Fagin, P. G. Kolaitis, R. J. Miller, and L. Popa. Data exchange: Semantics and query answering. In *ICDT 2003*, pages 207–224, 2003.

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

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

[10] M. Gebser, B. Kaufmann, A. Neumann, and T. Schaub. clasp: A conflict-driven answer set solver. In *In LPNMR '07*, pages 260–265. Springer, 2007.

[11] M. Gelfond and V. Lifschitz. Classical negation in logic programs and disjunctive databases. *New Generation Computing*, 9:365–385, 1991.

[12] D. S. Johnson and A. C. Klug. Testing containment of conjunctive queries under functional and inclusion dependencies. In *PODS*, pages 164–169, 1982.

[13] A. Klug. On conjunctive queries containing inequalities. *J. ACM*, 35(1):146–160, 1988.

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

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

[16] B. Marnette. Generalized schema-mappings: from termination to tractability. In *PODS*, pages 13–22, 2009.

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

[18] S. Paramonov. Query completeness—A logic programming approach. Technical Report KRDB13-2, KRDB Research Center, Free Univ. Bozen-Bolzano, 2013. http://www.inf.unibz.it/krdb/pub/tech-rep.php.

[19] S. Razniewski and W. Nutt. Checking query completeness over incomplete data. Technical Report KRDB11-2, KRDB Research Center, Free University of Bozen-Bolzano, 2011.

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

[21] O. Savković, M. Paramita, S. Paramonov, and W. Nutt. MAGIK: Managing completeness of data. In *CIKM*, pages 2725–2727, 2012.

[22] O. Savković, M. Paramita, A. Tomasi, and W. Nutt. Complete approximations of incomplete queries. *PVLDB*, 6(12):1378–1381, 2013.

[23] P. Simons, I. Niemelá, and T. Soininen. Extending and implementing the stable model semantics. *Artif. Intell.*, 138(1-2):181–234, June 2002.