Canonical Databases and Certain Answers under Key Constraints

Kilian Stoffel and Thomas Studer

IAM-04-009

November 2004

Canonical Databases and Certain Answers under Key Constraints

Kilian Stoffel Thomas Studer

Abstract

It is an important question to ask what can be deduced from a given view instance and knowledge about the relational schema with its constraints. We present a procedure which computes a minimal database that yields the given view instance and satisfies the constraints on the relational schema. It is minimal in the sense that it contains exactly the certain answers to queries asking for basic relations. In this paper, we consider only conjunctive view definitions and unique key constraints.

CR Categories: H.2.7 [Database Administration]: Security, Integrity, Protection; I.2.4 [Knowledge Representation Formalisms and Methods] Relation Systems
General Terms: Security
Additional Keywords: Data Privacy, Data Dependecies, Logical Inference

1 Introduction

This paper deals with the problem of constructing canonical database instances and computing certain answers in the presence of unique key constraints. The research presented here originates from the question of what can be deduced from a given set of answers to database queries assuming that one has knowledge about the database schema and the constraints defined on it.

Key constraints or *functional dependencies* are a fundamental concept in the theory and applications of relational databases [2, 13]. Such constraints do not only guarantee the consistency of the data stored in a database. They also provide (meta-)information about the relational schema and about the connections between different attributes occuring in the table definitions. On one hand this information is of an explanatory nature for the user. On the

other hand it can be employed by the database to perform computations and optimizations of queries, see for example [3] or for a more recent paper [9].
We are interested in how key constraints can be used to obtain knowledge about entries in a database which may not be directly accessed. Our aim is to prove that under given circumstances it is not possible to get such knowledge. This is important if there are restricitions on who may know what. Consider for example a database system storing patient data for a hospital. An accounting officer will have access to a view presenting him information about the costs of each patient's treatments. However, if this officer is not fully trusted, then one does not want him to know what treatment a patient received. He should only have access to the costs. Therefore we like to prove that he cannot infer information about the treatment from the data presented to him in the view instance and from knowledge about the relational schema and its constraints. Formally, if Q is a query asking for the treatment, then we want to prove that given the view instance, there is no *certain answer* to Q.
In recent years, computing certain answers has become an important tool in the study of relational databases. In the area of data integration and data warehousing for example, one is interested in answering queries using materialized views [1, 11]. A problem that formally can be treated by the notion of certain answer. Given a view instance I, a tuple t and a query Q, one likes to know whether t is in the answer to Q for all database instances yielding the view instance I. That is whether t is a certain answer to Q given I.
The addition of constraints to a relational database schema has some impact on the computation of certain answers [5, 6]. We investigate the following example. Let our schema consist of a table A with two attributes and a table G with one attribute. We define two views by the following queries

$$Q_1(x) \leftarrow A(x,y) \wedge G(y) \text{ and } Q_2(x) \leftarrow A(x,x),$$

respectively. Assume that the view instance consists of $Q_1(a), Q_2(a)$. This does not imply anything about the extension of G. The fact $Q_1(a)$ might stem from entries $A(a,b)$ and $G(b)$ and we do not know what b is. Hence, there is no way to say something about G except that it is non-empty. This situation completely changes if we add a unique key constraint to A. Assume that the first attribute of A is a unique key for this table. Then $Q_2(a)$ implies that $\forall y.(A(a,y) \rightarrow a = y)$. Therefore by $Q_1(a)$ we get that $G(a)$ must hold in any database which satisfies the key constraint and which yields the view instance.
In this paper we will deal with key constraints by the following formal con-

struction. The semantics of Q_1 is given by

$$Q_1^I := \{x \mid \exists y.((x,y) \in A^I \wedge y \in G^I)\}. \tag{1}$$

We see that the variable y is existentially quantified which hides the value assigned to it. In the presence of key constraints we can introduce a so-called *Skolem function* to get rid of this quantifier. In the expression $(x,y) \in A^I$ the value of y depends on x since x is in the position of a key attribute of A. Hence we can introduce a function f_A such that $f_A(x) = y$ for $(x,y) \in A^I$. (1) then becomes

$$Q_1^I := \{x \mid (x, f_A(x)) \in A^I \wedge f_A(x) \in G^I\}. \tag{2}$$

Therefore from $Q_1(a)$ we obtain

$$f_A(a) \in G^I. \tag{3}$$

The view defined by Q_2 tells us, how the function f_A is defined. We have

$$Q_2^I := \{x \mid (x, f_A(x)) \in A^I \wedge x = f_A(x)\}. \tag{4}$$

Hence $Q_2(a)$ implies $a = f_A(a)$. Together with (3) this yields $a \in G^I$.
One of the main technical problems in our research is that we have to deal with equalities. As noticed above, (4) implies $a = f_A(a)$. This means that we have to identify different terms with each other. Formally we tackle this problem by introducing an equivalence relation on the constants in the definition of a database instance.
The work we present here is related to the inverse rules algorithm[12, 7, 8] which generates query answering plans for information gathering agents. The key idea underlying that algorithm is to construct a set of rules that invert the view definitions. That is rules which show how to compute tuples for the database definitions from tuples of the views. We do a similar thing by computing the canonical database from a given view instance. Duschka and Levy [8] describe an inverse rules algorithm in the presence of key constraints. They also have to introduce an extra equivalence relation to identify different terms. This relation is defined by additional chase rules. However the inverse rules algorithm yields a maximally contained rewriting of a query that uses only relations that are actually stored in the information sources. We look for a method to define canonical database instances in order to compute certain answers.
Calì et al. [5] present a procedure to compute certain answers in the context of data integration under integrity constraints. However, in that context the problem to compute certain answers is to deal with foreign keys; whereas

we are interested in determining the consequences of having unique key constraints.
Finally, we have to mention that the main result of our paper follows also from recent research about data exchange. Fagin et al. [10] investigate a general setting for data exchange. They show that a canonical database can be computed in polynomial time and that it can be used to obtain the certain answers of conjunctive queries. However, we were not aware of that paper when we were developing our method.
Our paper is organized as follows. In the next section we introduce the formal framework of relational databases with unique key constraints. In Section 3 we show how to reformulate queries so that Skolem functions will take care of the constraints. Then we split the view instance into its atomic parts. Using these atomic facts we construct a canonical database instance in Section 4. This instance satisfies all the key constraints and yields the view instance. Finally, we show that this canonical database contains exactly the certain answers to queries asking about basic predicates. Section 5 then concludes the paper.

2 Framework

In this section we introduce the syntax and semantics of the relational database model with key constraints. We assume that the reader is familiar with the basic notions of relational databases [13].

A relational schema is given by a set of relation symbols and a set of key constraints, that are assertions on the relations symbols expressing conditions which must be satisfied by database instances. Formally, a *relational schema* $\mathcal{S}$ is a triple $(\mathcal{R}, \mathcal{K}, \Gamma)$ where:

- $\mathcal{R}$ is an alphabet of relation symbols $A, B, C, \ldots$. Each of them has an associated arity indicating the number of its attributes. To refer to a certain attribute we simply use the number corresponding to its position. Hence, the attributes of an n-ary relations are represented by the integers $1, \ldots, n$.

- $\mathcal{K}$ is a set of key constraint. A *key constraint* is an assertion of the form $\mathsf{key}(R) = i_1, \ldots, i_k$ where R is a relation symbol of $\mathcal{R}$ and $i_1, \ldots, i_k$ is a sequence of attributes of R. We assume that there is at most one key dependency specified for each relation in $\mathcal{R}$.

- Γ is a set of constants $a, b, c, \ldots$.

A *relational query* over a schema $\mathcal{S}$ is a formula to retrieve a set of tuples of constants from a database. We are only interested in conjunctive queries which are defined in the following way. An *atom* is an expression of the form $R(x_1, \ldots, x_n)$ where R is an n-ary relation symbol of $\mathcal{R}$ and $x_1, \ldots, x_n$ are either constants of Γ or variables. A *conjunctive query* Q of arity n is written in the form

$$Q(x_1, \ldots, x_n) \leftarrow \mathsf{conj}(x_1, \ldots, x_n, y_1, \ldots, y_m)$$

where

- Q belongs to a new alphabet of queries $\mathcal{Q}$ which is disjoint from $\mathcal{R}$ and
- $\mathsf{conj}(x_1, \ldots, x_n, y_1, \ldots, y_m)$ is a conjunction of atoms which are built from relation symbols from $\mathcal{R}$, constants from Γ and variables from the list $x_1, \ldots, x_n, y_1, \ldots, y_m$.

Let us now turn to the semantics. A *database instance* (or simply database) $\mathcal{B}$ for a schema $\mathcal{S} = (\mathcal{R}, \mathcal{K}, \Gamma)$ is a triple $(I, \Delta, \approx_{\mathcal{B}})$ where

- Δ is a set of constants so that $\Gamma \subset \Delta$,
- $\approx_{\mathcal{B}}$ is an equivalence relation on Δ, and
- I is a set of atoms of the form $R(a_1, \ldots, a_n)$ where R is an n-ary relation symbol of $\mathcal{R}$ and $(a_1, \ldots, a_n)$ is an n-tuple of constants of Δ.

Usually the definition of a database instance does not incorporate an equivalence relation on the alphabet. However, our canoncial database will be defined as kind of term model where certain terms have to be identified with each other. For instance, (4) implies that in the canonical database for that example, the term a will be equivalent to $f_A(a)$. This will be done in Definition 2 in the next section. To learn more about the general use of equivalence relations in the construction of term models, see [4].

In the following we will denote sequences of constants by $\vec{a}, \vec{b}, \vec{c}, \ldots$. Consequently sequences of variables will be abbreviated as $\vec{x}, \vec{y}, \vec{z}, \ldots$. The length of such sequences will be clear from the context. For $\vec{a} = a_1, \ldots, a_n$ and $\vec{b} = b_1, \ldots, b_n$ we write $\vec{a} \approx_{\mathcal{B}} \vec{b}$ if for all $1 \leq i \leq n$ we have $a_i \approx_{\mathcal{B}} b_i$. We will write $\exists \vec{x} \in \Delta$ for $\exists x_1 \in \Delta \ldots \exists x_n \in \Delta$. $\forall \vec{x} \in \Delta$ is used analogously.

By $R^{\mathcal{B}}$ we denote the set $\{\vec{x} \in \Delta \mid \exists \vec{y} \in \Delta.(\vec{x} \approx_{\mathcal{B}} \vec{y} \wedge R(\vec{y}) \in \mathcal{B})\}$. Note that we may have $a \in R^{\mathcal{B}}$ but $R(a) \notin I$.

A database $\mathcal{B}$ over $\mathcal{S}$ *satisfies* the key constraint $\mathsf{key}(R) = i_1, \ldots, i_k$ if for $\vec{a}, \vec{b} \in R^{\mathcal{B}}$ with $\vec{a} \not\approx_{\mathcal{B}} \vec{b}$ we have $\vec{a}[i_1, \ldots, i_k] \not\approx_{\mathcal{B}} \vec{b}[i_1, \ldots, i_k]$ where $\vec{a}[i_1, \ldots, i_k]$

is the projection of $\vec{a}$ over $i_1, \ldots, i_k$, for example $(a,b,c,d)[1,3]$ is (a,c). If $\mathcal{B}$ satisfies all constraints expressed by $\mathcal{K}$, then $\mathcal{B}$ is called *consistent* with respect to the schema $\mathcal{S} = (\mathcal{R}, \mathcal{K}, \Gamma)$. The set $\mathsf{sem}(\mathcal{S})$ is the set of all databases $\mathcal{B}$ that are consistent with $\mathcal{S}$.

Now we can give the semantics of queries. A query Q

$$Q(x_1, \ldots, x_n) \leftarrow \mathsf{conj}(x_1, \ldots, x_n, y_1, \ldots, y_m)$$

selects from a database instance $\mathcal{B} = (I, \Delta, \approx_{\mathcal{B}})$ an n-tuple $(a_1, \ldots, a_n)$ of constants of Δ. A tuple $(a_1, \ldots, a_n)$ is selected if there are constants $c_1, \ldots, c_m \in \Delta$ so that each atom of $\mathsf{conj}(a_1, \ldots, a_n, c_1, \ldots, c_m)$ is in $\mathcal{B}$. Formally, we define

$$\mathsf{ans}(Q, \mathcal{B}) := \{\vec{x} \in \Delta \mid \exists \vec{y} \in \Delta . \mathsf{conj}(\vec{x}, \vec{y}) \in \mathcal{B}\}$$

where $\mathsf{conj}(\vec{x}, \vec{y}) \in \mathcal{B}$ means that for each conjunct $A(\vec{z})$ of $\mathsf{conj}(\vec{x}, \vec{y})$ we have $z \in A^{\mathcal{B}}$. If R is a relation symbol of $\mathcal{R}$, then $\mathsf{ans}(R, \mathcal{B})$ is defined simply as $R^{\mathcal{B}}$.

An *answer set* $\mathcal{A}$ over an alphabet Λ is a set of facts $Q(\vec{t})$ where Q is a conjunctive query and $\vec{t} \in \Lambda$. We define

$$\mathsf{sem}(\mathcal{S}, \mathcal{A}) := \{\mathcal{B} \in \mathsf{sem}(\mathcal{S}) \mid \forall Q(\vec{t}) \in \mathcal{A} . \vec{t} \in \mathsf{ans}(Q, \mathcal{B})\}.$$

This means that $\mathsf{sem}(\mathcal{S}, \mathcal{A})$ is the set of all consistent databases for $\mathcal{S}$ that yield at least the answers given by $\mathcal{A}$. We call $\mathsf{sem}(\mathcal{S}, \mathcal{A})$ the *semantics* of $\mathcal{S}$ with respect to the answer set $\mathcal{A}$.

An answer set $\mathcal{A}$ may be regarded as a *view instance*. The views are defined by the answers to the queries in $\mathcal{A}$. If for a tuple $\vec{t}$ we find $Q(\vec{t}) \in \mathcal{A}$, then the view Q contains $\vec{t}$.

For $Q \in \mathcal{R} \cup \mathcal{Q}$ we define the set of *certain answers* to Q by

$$\mathsf{cert}(Q, \mathcal{S}, \mathcal{A}) := \bigcap_{\mathcal{B} \in \mathsf{sem}(\mathcal{S}, \mathcal{A})} \mathsf{ans}(Q, \mathcal{B}).$$

That is the set $\mathsf{cert}(Q, \mathcal{S}, \mathcal{A})$ consists of the tuples which are answers to the query Q in every database in the semantics of $\mathcal{S}$ with respect to $\mathcal{A}$.

We have $\vec{t} \in \mathsf{cert}(Q, \mathcal{S}, \mathcal{A})$ if every database instance over $\mathcal{S}$ which yields the view instance $\mathcal{A}$ answers $\vec{t}$ to the query Q. Note that this reading of cert adopts an *open world assumption*. We might know only a part of the extension of the view instance. Hence our definition of certain answer considers also databases that yield bigger view instances than $\mathcal{A}$.

Using these definitions we obtain for a relation G of a schema $\mathcal{S}$ that G is safe with respect to a given answer set $\mathcal{A}$ if $\mathsf{cert}(G, \mathcal{S}, \mathcal{A}) = \emptyset$ holds. In this case, we cannot get any information about the extension of G. That is whether any given constant belongs to G or not.

3 Query reformulation

In this section we show how to rewrite queries so that we can build the canonical database instance.
First we introduce symbols for the Skolem functions that deal with the functional dependencies. Starting from an initial alphabet Λ of constants, we define a new alphabet $\Lambda_{\mathcal{S}}$ of the terms built up from the constants of Λ, variables $x, y, z, \ldots$ and the Skolem functions.

Definition 1. Let $\mathcal{S} = (\mathcal{R}, \mathcal{K}, \Gamma)$ be a database schema and let $\Lambda \supset \Gamma$ be an alphabet of constants. We define a new alphabet $\Lambda_{\mathcal{S}}$ and a function depth mapping elements of $\Lambda_{\mathcal{S}}$ to natural numbers as follows.

- $\Lambda \subset \Lambda_{\mathcal{S}}$ and for every constant $a \in \Lambda$ we set $\mathsf{depth}(a) := 0$.
- $\Lambda_{\mathcal{S}}$ contains infinitely many variables $x, y, z, \ldots$ (possibly with subscripts). We define $\mathsf{depth}(x) := 0$ for any variable x.
- For each n-ary relation symbol A in $\mathcal{R}$ with $key(A) = i_1, \ldots, i_k \in \mathcal{K}$ we introduce new function symbols f_{A,i_l} for $i_l \notin \mathsf{key}(A)$ and $1 \leq i_l \leq n$. For every such function symbol and arbitrary terms $r_1, \ldots, r_k \in \Lambda_{\mathcal{S}}$ with $\mathsf{depth}(r_h) = 0$ $(1 \leq h \leq k)$ we set $f_{A,i_l}(r_1, \ldots, r_k) \in \Lambda_{\mathcal{S}}$ and $\mathsf{depth}(f_{A,i_l}(r_1, \ldots, r_k)) := 1$.

Now we are going to build the foundations for our canonical database instance $\mathcal{B}$. Depending on a given answer set $\mathcal{A}$ over an alphabet Λ we will compute a set $O_{\mathcal{A}}$ of $\Lambda_{\mathcal{S}}$ atoms and the extension of a binary relation $\mathsf{Eq}_{\mathcal{A}}$ on $\Lambda_{\mathcal{S}}$. The alphabet $\Lambda_{\mathcal{S}}$ will be the set of constants of the canonical database $\mathcal{B}$. The set $O_{\mathcal{A}}$ will be the collection of all atoms of $\mathcal{B}$. The relation $\mathsf{Eq}_{\mathcal{A}}$ will serve as starting point for the definition of the equivalence relation $\approx_{\mathcal{B}}$ used in $\mathcal{B}$. Consider the example given in the introductory section. From (2) and (4) we will get $A(a, f_A(a)) \in O_{\mathcal{A}}$, $G(f_A(a)) \in O_{\mathcal{A}}$ and $(a, f_A(a)) \in \mathsf{Eq}_{\mathcal{A}}$. Hence we will have $a \approx_{\mathcal{B}} f_A(a)$ which leads us to a database satisfying $G(a)$.
The construction of $O_{\mathcal{A}}$ and $\mathsf{Eq}_{\mathcal{A}}$ takes three steps for each element of $\mathcal{A}$. First the queries for which an answer is provided in $\mathcal{A}$ are rectified so that no unintended interaction between them can occurr. Then Skolem functions are introduced in the bodies of the queries. The conjuncts of these bodies build the set $O_{\mathcal{A}}$. Each replacemt of a term by one containing a Skolem function is recorded in $\mathsf{Eq}_{\mathcal{A}}$. Finally the actual answers to the queries are worked into $O_{\mathcal{A}}$ and $\mathsf{Eq}_{\mathcal{A}}$.

Definition 2. Let $Q(t_1, \ldots, t_n) \in \mathcal{A}$ and assume Q is the query

$$Q(x_1, \ldots, x_n) \leftarrow \mathsf{conj}(x_1, \ldots, x_n, y_1, \ldots, y_m).$$

Then we define the sets $Q(t_1, \dots, t_n)^*$ and $Q(t_1, \dots, t_n)^{\mathsf{Eq}}$ by the following procedure where $\vec{t}$ denotes the sequence $t_1, \dots, t_n$.

1. We index all variables in the body of the query by $Q, \vec{t}$. Hence, we obtain $\mathsf{conj}(x_{1,Q,\vec{t}}, \dots, x_{n,Q,\vec{t}}, y_{1,Q,\vec{t}}, \dots, y_{m,Q,\vec{t}})$ which will be abbreviated as conj'.

2. For all $A(\vec{z})$ in conj' with $key(A) = i_1, \dots, i_k \in \mathcal{K}$ and for all $j \notin \mathsf{key}(A)$, we replace the variable or constant z_j at attribute position j in $A(\vec{z})$ by $f_{A,j}(\vec{z}[\mathsf{key}(A)])$. We call the resulting atom $A(\vec{z})^*$ and add it to $Q(t_1, \dots, t_n)^*$. Further we include the pair $(z_j, f_{A,j}(\vec{z}[\mathsf{key}(A)]))$ to $Q(t_1, \dots, t_n)^{\mathsf{Eq}}$. Note that z_j may be one of the variables of conj' or a constant of Γ.

3. We replace the variable $x_{i,Q,\vec{t}}$ by t_i in all elements of $Q(t_1, \dots, t_n)^*$ and $Q(t_1, \dots, t_n)^{\mathsf{Eq}}$ for all $1 \leq i \leq n$.

We define the set $O_{\mathcal{A}}$ and the relation $\mathsf{Eq}_{\mathcal{A}}$ by

$$O_{\mathcal{A}} := \bigcup_{Q(\vec{t}) \in \mathcal{A}} Q(\vec{t})^* \text{ and } \mathsf{Eq}_{\mathcal{A}}(a, b) :\Leftrightarrow (a, b) \in \bigcup_{Q(\vec{t}) \in \mathcal{A}} Q(\vec{t})^{\mathsf{Eq}}.$$

Let us look at two examples to see the previous definition in action.

Example 3. Let Γ contain the constants a, b. Further, let $\mathcal{R}$ contain a binary relation R, a ternary relation S and a unary relation G. $\mathcal{K}$ contains the following two key constraint $\mathsf{key}(R) = 1$ and $\mathsf{key}(S) = 1, 2$. Hence in $\Lambda_{\mathcal{S}}$ we introduce a function symbol $f_{R,2}$ and a symbol $f_{S,3}$. We look at the following three queries.

- $O(x) \leftarrow R(x, x)$
- $P(x) \leftarrow R(x, y) \wedge S(b, y, z) \wedge G(z)$
- $Q(x) \leftarrow R(x, y) \wedge S(y, a, y)$

Assume $\mathcal{A}$ consists of $O(a), O(b), P(a), Q(b)$. We obtain the following after rectifying the rules and introducing Skolem functions.

- $R(x_{O,a}, f_{R,2}(x_{O,a})) \in O(a)^*$ and $(x_{O,a}, f_{R,2}(x_{O,a})) \in O(a)^{\mathsf{Eq}}$,
- $R(x_{O,b}, f_{R,2}(x_{O,b})) \in O(b)^*$ and $(x_{O,b}, f_{R,2}(x_{O,b})) \in O(b)^{\mathsf{Eq}}$,
- $R(x_{P,a}, f_{R,2}(x_{P,a})), S(b, y_{P,a}, f_{S,3}(b, y_{P,a})), G(z_{P,a}) \in P(a)^*$ and $(y_{P,a}, f_{R,2}(x_{P,a})), (z_{P,a}, f_{S,3}(b, y_{P,a})) \in P(a)^{\mathsf{Eq}}$,

- $R(x_{Q,b}, f_{R,2}(x_{Q,b})), S(y_{Q,b}, a, f_{S,3}(y_{Q,b}, a)) \in Q(b)^*$ and $(y_{Q,b}, f_{R,2}(x_{Q,b})), (y_{Q,b}, f_{S,3}(y_{Q,b}, a)) \in Q(b)^{\mathsf{Eq}}$.

In the third step we replace $x_{O,a}$ by a, $x_{O,b}$ by b, $x_{P,a}$ by a and $x_{Q,b}$ by b. Hence we get

- $R(a, f_{R,2}(a)) \in O(a)^*$ and $(a, f_{R,2}(a)) \in O(a)^{\mathsf{Eq}}$,
- $R(b, f_{R,2}(b)) \in O(b)^*$ and $(b, f_{R,2}(b)) \in O(b)^{\mathsf{Eq}}$,
- $R(a, f_{R,2}(a)), S(b, y_{P,a}, f_{S,3}(b, y_{P,a})), G(z_{P,a}) \in P(a)^*$ and $(y_{P,a}, f_{R,2}(a)), (z_{P,a}, f_{S,3}(b, y_{P,a})) \in P(a)^{\mathsf{Eq}}$,
- $R(b, f_{R,2}(b)), S(y_{Q,b}, a, f_{S,3}(y_{Q,b}, a)) \in Q(b)^*$ and $(y_{Q,b}, f_{R,2}(b)), (y_{Q,b}, f_{S,3}(y_{Q,b}, a)) \in Q(b)^{\mathsf{Eq}}$.

Example 4. It can happen that a variable appears in two positions where it has to be substituted by Skolem functions. Then the algorithm of Definition 2 makes both replacments and introduces two pairs (one for each replacement) into the $\mathsf{Eq}_{\mathcal{A}}$ relation. Look at the query

$$Q(x) \leftarrow R(x, y) \wedge S(a, y),$$

where R and S are binary relations with a unique key constraint on the first attribute. We note that the variable y appears twice at positions where it depends on key attributes. Let $Q(b) \in \mathcal{A}$. Then we get first

$$R(x_{Q,b}, f_{R,2}(x_{Q,b})), S(a, f_{S,2}(a)) \in Q(b)^*$$

and

$$(y_{Q,b}, f_{R,2}(x_{Q,b})), (y_{Q,b}, f_{S,2}(a)) \in Q(b)^{\mathsf{Eq}}$$

In the third step we replace $x_{Q,b}$ by b. This yields

$$R(b, f_{R,2}(b)), S(a, f_{S,2}(a)) \in Q(b)^*$$

and

$$(y_{Q,b}, f_{R,2}(b)), (y_{Q,b}, f_{S,2}(a)) \in Q(b)^{\mathsf{Eq}}.$$

4 Computing Certain Answers

Now we are prepared to define the canonical database and show that it contains exactly the certain answers to queries asking for basic predicates. First we build the closure of $\mathsf{Eq}_{\mathcal{A}}$ under transitivity and applications of functions. Using the resulting relation $\approx_{\mathsf{Eq}_{\mathcal{A}}}$, we can define the canonical database $\mathcal{B}$ as $(O_{\mathcal{A}}, \Gamma_{\mathcal{S}}, \approx_{\mathsf{Eq}_{\mathcal{A}}})$. This database instance is consistent with respect to the schema $\mathcal{S}$ and yields the answer set $\mathcal{A}$. Further, we prove that for any relation symbol G we have

$$\vec{t} \in \mathsf{cert}(G, \mathcal{S}, \mathcal{A}) \text{ implies } \vec{t} \in G^{\mathcal{B}}. \tag{5}$$

To show the reverse direction of this statement, we introduce sets of functions $\mathsf{KeyFun}(A, i, \mathcal{B}, \mathcal{S})$ which can handle the dependencies given by the key constraints. With those functions we can define a mapping, called valuation, from the extendend language $\Gamma_{\mathcal{S}}$ resulting from the rewriting of the queries to the the original set of constants Γ. Hence we can apply such a mapping to the canonical database we have defined above in order to obtain results about database instances over Γ. It is shown that any valuation respecting the answerset $\mathcal{A}$ maps terms $\vec{t} \in \Gamma_{\mathcal{S}}$ with $G(\vec{t}) \in O_{\mathcal{A}}$ to constants $\vec{s} \in \Gamma$ such that $\vec{s} \in \mathsf{cert}(G, \mathcal{S}, \mathcal{A})$. This statement is the reverse direction of (5) we are looking for.

Definition 5. Let $\mathcal{A}$ be an answerset over the alphabet Γ. Depending on $\mathsf{Eq}_{\mathcal{A}}$, we inductively define a relation $\approx_{\mathsf{Eq}_{\mathcal{A}}}$ on $\Gamma_{\mathcal{S}}$ by the following rules.

- If $r \in \Gamma_{\mathcal{S}}$, then $r \approx_{\mathsf{Eq}_{\mathcal{A}}} r$ (Ref).
- If $(r, s) \in \mathsf{Eq}_{\mathcal{A}}$, then $r \approx_{\mathsf{Eq}_{\mathcal{A}}} s$ (Taut).
- If $s \approx_{\mathsf{Eq}_{\mathcal{A}}} r$, then $r \approx_{\mathsf{Eq}_{\mathcal{A}}} s$ (Sym).
- If $r \approx_{\mathsf{Eq}_{\mathcal{A}}} t$ and either $(t, s) \in \mathsf{Eq}_{\mathcal{A}}$ or $(s, t) \in \mathsf{Eq}_{\mathcal{A}}$, then $r \approx_{\mathsf{Eq}_{\mathcal{A}}} s$ (Trans).
- If $r \approx_{\mathsf{Eq}_{\mathcal{A}}} s$ with $\mathsf{depth}(r) = 0$ and $\mathsf{depth}(s) = 0$, then $f(r) \approx_{\mathsf{Eq}_{\mathcal{A}}} f(s)$ for $f(r), f(s) \in \Gamma_{\mathcal{S}}$ (Funct).

In the sequel we generally assume that $\mathcal{A}$ is finite and therefore $\mathsf{Eq}_{\mathcal{A}}$ is finite, too. In this case the rules given above define an algorithm to decide whether $r \approx_{\mathsf{Eq}_{\mathcal{A}}} s$ holds for $r, s \in \Gamma_{\mathcal{S}}$. Starting from $r \approx_{\mathsf{Eq}_{\mathcal{A}}} s$ we can do a backward proof search, testing which rules may have been applied. The two critical cases are applications of the (Trans) and of the (Sym) rules. However, since $\mathsf{Eq}_{\mathcal{A}}$ is a finite relation, there are only finitely many instantiations of the

(Trans) rule. To cope with the (Sym) rule, we can do a simple loop check in the decision procedure.
To simplify our notation, we will drop the subscript $\mathsf{Eq}_{\mathcal{A}}$. No confusion should arise since we only deal with one $\mathsf{Eq}_{\mathcal{A}}$ relation.
By the following lemma we know that $\approx$ is an equivalence relation. Hence, it can be employed in the definition of database instances.

Lemma 6. *If $r \approx t$ and $t \approx s$, then $r \approx s$.*

Proof. We show that if $r \approx t$ and either $t \approx s$ or $s \approx t$, then $r \approx s$. This is proven by induction on the length of the derivations of $t \approx s$ and $s \approx t$. □

Theorem 7. *Let $\mathcal{A}$ be an answerset over the alphabet Γ and let $\mathcal{B}$ be the database instance $(O_{\mathcal{A}}, \Gamma_{\mathcal{S}}, \approx)$ over the schema $\mathcal{S} = (\mathcal{R}, \mathcal{K}, \Gamma)$. Then we have $\mathcal{B} \in \mathsf{sem}(\mathcal{S})$.*

Proof. We have to show that all key constraints in $\mathcal{S}$ are satisfied. For simplicity we deal with the following exemplary case. Assume A is a binary relation of $\mathcal{R}$ and $\mathsf{key}(A) = 1 \in \mathcal{K}$. Let $a_1, a_2, b_1, b_2 \in \Gamma_{\mathcal{S}}$ with $(a_1, a_2) \in A^{\mathcal{B}}$, $(b_1, b_2) \in A^{\mathcal{B}}$ and $a_1 \approx b_1$. We have to show $a_2 \approx b_2$. There are terms a_1', a_2', b_1', b_2' such that $a_1' \approx a_1, a_2' \approx a_2, b_1' \approx b_1, b_2' \approx b_2$ as well as

$$A(a_1', a_2') \in O_{\mathcal{A}} \text{ and } A(b_1', b_2') \in O_{\mathcal{A}}.$$

By the construction of $O_{\mathcal{A}}$ we know that $\mathsf{depth}(a_1') = 0$ and $\mathsf{depth}(b_1') = 0$. Further, we know $a_2' \equiv f_{A,2}(a_1')$ and $b_2' \equiv f_{A,2}(b_1')$. Therefore from

$$a_1' \approx a_1 \approx b_1 \approx b_1'$$

we obtain by Definition 5 that

$$a_2 \approx a_2' \equiv f_{A,2}(a_1') \approx f_{A,2}(b_1') \equiv b_2' \approx b_2$$

which finishes the proof. □

Theorem 8. *Let $\mathcal{A}$ be an answerset over the alphabet Γ and let $\mathcal{B}$ be the database instance $(O_{\mathcal{A}}, \Gamma_{\mathcal{S}}, \approx)$ over the schema $\mathcal{S} = (\mathcal{R}, \mathcal{K}, \Gamma)$. We find that*

1. *for any $Q(\vec{a}) \in \mathcal{A}$ we have $\vec{a} \in \mathsf{ans}(Q, \mathcal{B})$ and*
2. *$\mathcal{B} \in \mathsf{sem}(\mathcal{S}, \mathcal{A})$.*

Proof. The first claim is shown as follows. Let Q be the query

$$Q(\vec{x}) \leftarrow \mathsf{conj}(\vec{x}, \vec{y}).$$

Hence, to show $\vec{a} \in \mathsf{ans}(Q, \mathcal{B})$ means to prove

$$\exists \vec{y} \in \Gamma_{\mathcal{S}}.\mathsf{conj}(\vec{a}, \vec{y}) \in \mathcal{B}. \tag{6}$$

We investigate the following exemplary case. Assume $\mathsf{conj}(\vec{a}, \vec{y})$ consists of one atom $A(s_1, s_2)$ where A is a binary relation symbol with the constraint $\mathsf{key}(A) = 1$. We distinguish the different possibilities for s_1 and s_2.

- Assume s_1 is a constant of $\vec{a}$ and s_2 is a variable of $\vec{y}$. Set $t_2 :\equiv f_{A,2}(s_1)$. We immediately get $A(s_1, t_2) \in O_{\mathcal{A}}$.
- Assume $s_1 \equiv y_i$ and s_2 also is a variable of $\vec{y}$. Set $t_1 :\equiv y_{i,Q,\vec{a}}$ and set $t_2 :\equiv f_{A,2}(t_1)$. We immediately get $A(t_1, t_2) \in O_{\mathcal{A}}$.
- Assume s_1 and s_2 are both constants of $\vec{a}$. Set $t_2 :\equiv f_{A,2}(s_1)$. We get $A(s_1, t_2) \in I$ as well as $\mathsf{Eq}_{\mathcal{A}}(s_2, t_2)$. Hence we obtain $s_2 \approx t_2$ and may conclude $(s_1, s_2) \in A^{\mathcal{B}}$.
- Assume $s_1 \equiv y_i$ and s_2 is a constant of $\vec{a}$. Set $t_1 :\equiv y_{i,Q,\vec{a}}$ and set $t_2 :\equiv f_{A,2}(t_1)$. We get $A(t_1, t_2) \in I$ as well as $\mathsf{Eq}_{\mathcal{A}}(s_2, t_2)$. Hence we obtain $s_2 \approx t_2$ and infer $(t_1, s_2) \in A^{\mathcal{B}}$.

These four cases cover all possibilities. Therefore we conclude that (6) holds. The second claim is an direct consequence of the first claim and the previous theorem. □

From the second claim of this theorem we immediately get the following corollary by the definition of $\mathsf{cert}(G, \mathcal{S}, \mathcal{A})$.

Corollary 9. *Assume $\mathcal{S} = (\mathcal{R}, \mathcal{K}, \Gamma)$ is a relational schema and $\mathcal{A}$ is an answerset over Γ. Let $\mathcal{B}$ be the database instance $(O_{\mathcal{A}}, \Gamma_{\mathcal{S}}, \approx)$, let G be a relation symbol of $\mathcal{R}$ and let $\vec{t} \in \Gamma$. Then $\vec{t} \in \mathsf{cert}(G, \mathcal{S}, \mathcal{A})$ implies $\vec{t} \in G^{\mathcal{B}}$.*

Now we are going to show the reverse direction of this corollary.

Let $\mathcal{B}$ be a database instance which is consistent with the relational schema $\mathcal{S} = (\mathcal{R}, \mathcal{K}, \Gamma)$, that is $\mathcal{B} \in \mathsf{sem}(\mathcal{S})$. We define sets of functions $\mathsf{KeyFun}(A, i_l, \mathcal{B}, \mathcal{S})$ that will take care of the key constraints.

Definition 10. Let $\mathcal{S} = (\mathcal{R}, \mathcal{K}, \Gamma)$ be a database schema and $\mathcal{B} = (I, \Delta, \approx_{\mathcal{B}})$ a consistent database instance over it. For each n-ary relation symbol A in $\mathcal{R}$ with $key(A) = i_1, \ldots, i_k \in \mathcal{K}$ and each $i_l \notin \mathsf{key}(A)$ we introduce a set $\mathsf{KeyFun}(A, i_l, \mathcal{B}, \mathcal{S})$ of k-ary functions as follows. A function $f : \Delta^k \to \Delta$ belongs to $\mathsf{KeyFun}(A, i_l, \mathcal{B}, \mathcal{S})$ if

- for all $A(x_1, \ldots, x_n) \in I$ we have $f(x_{i_1}, \ldots, x_{i_k}) \approx_{\mathcal{B}} x_{i_l}$ and

- $\forall \vec{x}, \vec{y} \in \Delta.(\vec{x} \approx_{\mathcal{B}} \vec{y} \rightarrow f(\vec{x}) \approx_{\mathcal{B}} f(\vec{y}))$.

Observe that $\mathsf{KeyFun}(A, i_l, \mathcal{B}, \mathcal{S})$ is defined exactly for those A and i_l for which we introduced function symbols f_{A,i_l} in the extended language $\Gamma_{\mathcal{S}}$ (see Definition 1).
The sets $\mathsf{KeyFun}(A, i_l, \mathcal{B}, \mathcal{S})$ are well-defined. That is they are not empty since $\mathcal{B}$ satisfies all key constraints in $\mathcal{S}$. We have the following lemma.

Lemma 11. *Let $\mathcal{B} = (I, \Delta, \approx_{\mathcal{B}})$ be a consistent database instance over the schema $\mathcal{S}$. Assume A is a relation symbol of $\mathcal{S}$ with a key constraint defined on it. Then we have for $i \notin \mathsf{key}(A)$ that $\mathsf{KeyFun}(A, i, \mathcal{B}, \mathcal{S}) \neq \emptyset$.*

Proof. Let us only investigate the case where A is a binary relation symbol with $\mathsf{key}(A) = 1$. We show that $\mathsf{KeyFun}(A, 2, \mathcal{B}, \mathcal{S}) \neq \emptyset$. We define a function f as follows. Assume $a \in \Delta$.

- If there exist $a', b \in \Delta$ so that $a \approx_{\mathcal{B}} a'$ and

$$A(a', b) \in I, \tag{7}$$

  then choose one of the b so that (7) holds and set $f(a) = b$.

- If no such $a', b \in \Delta$ exist, then we define $f(a) = a$.

We find that the so defined function f satisfies both conditions of Definition 10.

- Assume $A(x, y) \in I$. Then $f(x) = b$ for some $b, x' \in \Delta$ with $A(x', b) \in I$ and $x \approx_{\mathcal{B}} x'$. Since $\mathcal{B}$ satisfies the key constraints of $\mathcal{S}$, we get $y \approx_{\mathcal{B}} b$ and hence $f(x) \approx_{\mathcal{B}} y$.

- Let $x, y \in \Delta$ with $x \approx_{\mathcal{B}} y$. If there exist $x', b \in \Delta$ with $A(x', b) \in I$ and $x \approx_{\mathcal{B}} x'$, then we have $f(x) = c$ for some $A(x'', c) \in I$ with $x \approx_{\mathcal{B}} x''$. Moreover we get $f(y) = d$ for some $A(y'', d) \in I$ with $y \approx_{\mathcal{B}} y''$. By $x \approx_{\mathcal{B}} y$ and transitivity of $\approx_{\mathcal{B}}$ we find $x'' \approx_{\mathcal{B}} y''$. Therefore

$$f(x) = c \approx_{\mathcal{B}} d = f(y)$$

  since $\mathcal{B}$ satisfies the key constraints of $\mathcal{S}$.

Hence we conclude $f \in \mathsf{KeyFun}(A, 2, \mathcal{B}, \mathcal{S})$. □

Definition 12. A *valuation* v for a database instance $\mathcal{B} = (I, \Gamma, \approx_{\mathcal{B}})$ over a schema $\mathcal{S}$ is a function that maps $\Gamma_{\mathcal{S}}$ to Γ satisfying the following.

- For any constant $a \in \Gamma_{\mathcal{S}}$ we have $\mathsf{v}(a) = a$.

- For any variable $x \in \Gamma_{\mathcal{S}}$, there exists a constant $a \in \Gamma$ with $\mathsf{v}(x) = a$.
- Each function symbol $f_{A,i}$ introduced in $\Gamma_{\mathcal{S}}$ is mapped to a function $\mathsf{v}(f_{A,i}) \in \mathsf{KeyFun}(A, i, \mathcal{B}, \mathcal{S})$. Then we define

$$\mathsf{v}(f_{A,i}(r_1, \ldots, r_n)) = \mathsf{v}(f_{A,i})(\mathsf{v}(r_1), \ldots, \mathsf{v}(r_n)).$$

Definition 13. Let $\mathcal{B} = (I, \Gamma, \approx_{\mathcal{B}})$ be a database instance.

- For $r, s \in \Gamma_{\mathcal{S}}$ we define that $\mathcal{B}, \mathsf{v} \vdash r \approx_{\mathcal{B}} s$ holds if $\mathsf{v}(r) \approx_{\mathcal{B}} \mathsf{v}(s)$.
- For $r_1, \ldots, r_n \in \Gamma_{\mathcal{S}}$ we define that $\mathcal{B}, \mathsf{v} \vdash G(r_1, \ldots, r_n)$ holds if

$$G(\mathsf{v}(r_1), \ldots, \mathsf{v}(r_n)) \in I.$$

- Let $\mathsf{conj}(\vec{a}, \vec{y})$ contain only terms of $\Gamma_{\mathcal{S}}$. We define that $\mathcal{B}, \mathsf{v} \vdash \mathsf{conj}(\vec{a}, \vec{y})$ holds if $\mathcal{B}, \mathsf{v} \vdash A(\vec{r})$ holds for all atoms $A(\vec{r})$ occurring in $\mathsf{conj}(\vec{a}, \vec{y})$.

We will make use of the following notation: if $\vec{r} = r_1, \ldots, r_n$ is a sequence of elemtents of $\Gamma_{\mathcal{S}}$, then $\mathsf{v}(\vec{r})$ denotes the sequence $\mathsf{v}(r_1), \ldots, \mathsf{v}(r_n)$.

Definition 14. Assume $\mathcal{S} = (\mathcal{R}, \mathcal{K}, \Gamma)$ is a relational schema and $\mathcal{A}$ is an answerset over an alphabet Γ. Let $\mathcal{B} \in \mathsf{sem}(\mathcal{S}, \mathcal{A})$. We say a *valuation* v *for* $\mathcal{B}$ *satisfies* $\mathcal{A}$ if for all queries $Q(\vec{x}) \leftarrow \mathsf{conj}(\vec{x}, \vec{y})$ with $Q(\vec{r}) \in \mathcal{A}$ we have $\mathcal{B}, \mathsf{v} \vdash \mathsf{conj}(\vec{r}, \vec{y}_{Q,r})$. The term $\vec{y}_{Q,\vec{r}}$ denotes the result of adding the subscript $Q, \vec{r}$ to each variable of $\vec{y}$ as it is done in the first step of Definition 2.

Lemma 15. *Let* $\mathcal{B} \in \mathsf{sem}(\mathcal{S}, \mathcal{A})$. *Then there exists a valuation* v *for* $\mathcal{B}$ *satisfying* $\mathcal{A}$.

Proof. For all queries $Q(\vec{x}) \leftarrow \mathsf{conj}(\vec{x}, \vec{y})$ with $Q(\vec{r}) \in \mathcal{A}$ the valuation v must satisfy $\mathcal{B}, \mathsf{v} \vdash \mathsf{conj}(\vec{r}, \vec{y}_{Q,r})$. This can be achieved by defining v as follows.

- The valuation v maps any constant $\mathcal{B}$ to itself.
- $\mathcal{B} \in \mathsf{sem}(\mathcal{S}, \mathcal{A})$ means that for each $Q(\vec{r}) \in \mathcal{A}$ there are constants $\vec{a}$ so that $\mathsf{conj}(\vec{r}, \vec{a}) \in \mathcal{B}$. We define the v-image of the variable $y_{i,Q,\vec{r}}$ to be the corresponding a_i.
- For each $f_{A,i} \in \Gamma_{\mathcal{S}}$ we define $\mathsf{v}(f_{A,i})$ to be an arbitrary element of $\mathsf{KeyFun}(A, i, \mathcal{B}, \mathcal{S})$. This can be done since by Lemma 11 this set is non-empty.

□

Lemma 16. *Assume $\mathcal{S} = (\mathcal{R}, \mathcal{K}, \Gamma)$ is a relational schema and $\mathcal{A}$ is an answerset over Γ. Let $\mathcal{B} \in \mathsf{sem}(\mathcal{S}, \mathcal{A})$. Let $\mathsf{Eq}_{\mathcal{A}}$ be defined as above. If v is a valuation for $\mathcal{B}$ satisfying $\mathcal{A}$, then we have*

$$(a, b) \in \mathsf{Eq}_{\mathcal{A}} \Longrightarrow \mathcal{B}, \mathsf{v} \vdash a \approx_{\mathcal{B}} b$$

Proof. $(a, b) \in \mathsf{Eq}_{\mathcal{A}}$ implies that b is of the form $f_{A,i}(\vec{z})$. For simplicity we just investigate the case where A is a binary relation with $key(A) = 1$. Assume $(a, f_{A,2}(z)) \in \mathsf{Eq}_{\mathcal{A}}$. We have to show that for any $f \in \mathsf{KeyFun}(A, 2, \mathcal{B}, \mathcal{S})$ the following holds:

$$\mathsf{v}(a) \approx_{\mathcal{B}} f(\mathsf{v}(z)). \tag{8}$$

We argue as follows: $(a, f_{A,2}(z)) \in \mathsf{Eq}_{\mathcal{A}}$ implies that there exists a query $Q(\vec{x}) \leftarrow \mathsf{conj}(\vec{x}, \vec{y})$ with $Q(\vec{r}) \in \mathcal{A}$ so that

$$(a, f_{A,2}(z)) \in Q(\vec{r})^{\mathsf{Eq}} \text{ and } A(z, a) \in \mathsf{conj}(\vec{r}, \vec{y}_{Q,\vec{r}}).$$

Since v satisfies $\mathcal{A}$, we know $\mathcal{B}, \mathsf{v} \vdash A(z, a)$. This is $A(\mathsf{v}(z), \mathsf{v}(a)) \in I$. Therefore by the definition of $\mathsf{KeyFun}(A, 2, \mathcal{B}, \mathcal{S})$ we conclude that (8) holds. □

The next lemma states that a valuation for a database instance respects the equivalence relation used in the definition of the database.

Lemma 17. *Assume $\mathcal{S} = (\mathcal{R}, \mathcal{K}, \Gamma)$ is a relational schema and $\mathcal{A}$ is an answerset over Γ. Let $\approx$ be the relation of Definition 5. Assume we are given two elements s, t of $\Gamma_{\mathcal{S}}$ with $s \approx t$. Let $\mathcal{B} = (I, \Gamma, \approx_{\mathcal{B}})$ be an arbitrary database instance in $\mathsf{sem}(\mathcal{S}, \mathcal{A})$ and v a valuation for $\mathcal{B}$ satisfying $\mathcal{A}$. Then we have $\mathcal{B}, \mathsf{v} \vdash s \approx_{\mathcal{B}} t$.*

Proof. We prove the statement by induction on the length of the derivation of $s \approx t$. The possible cases for the last rule that has been applied in the derivation are the following:

- (Ref): Trivial since $\approx_{\mathcal{B}}$ is an equivalence relation by definition.
- (Taut): By Lemma 16.
- (Sym): The claim follows from the induction hypothesis and the symmetry of $\approx_{\mathcal{B}}$.
- (Trans): The claim follows from the induction hypothesis and the transistivity of $\approx_{\mathcal{B}}$.
- (Funct): The claim follows from the induction hypothesis and the fact that the functions of $\mathsf{KeyFun}(A, i, \mathcal{B}, \mathcal{S})$ respect $\approx_{\mathcal{B}}$ by definition.

□

The following lemma says that valuations for databases are well-behaved with respect to the atoms of these database instances.

Lemma 18. *Let $\mathcal{B} = (I, \Gamma, \approx_{\mathcal{B}})$ be a database instance and v a valuation for it. Let $Q(\vec{x}) \leftarrow \mathsf{conj}(\vec{x}, \vec{y})$ be a query containing the relation symbol G. Then we have*

$$G(\vec{t}) \in Q(\vec{r})^* \ \wedge \ \mathcal{B}, \mathsf{v} \vdash \mathsf{conj}(\vec{r}, \vec{y}_{Q,\vec{r}}) \implies \mathcal{B}, \mathsf{v} \vdash G(\vec{t}).$$

Proof. For simplicity we just look at the case where G is a binary relation with $\mathsf{key}(G) = 1$. Assume $G(t_1, f_{G,2}(t_1)) \in Q(\vec{r})^*$. Then there is a t_2 so that $G(t_1, t_2)$ is contained in $\mathsf{conj}(\vec{r}, \vec{y}_{Q,\vec{r}})$. Hence $\mathcal{B}, \mathsf{v} \vdash G(t_1, t_2)$. This is

$$G(\mathsf{v}(t_1), \mathsf{v}(t_2)) \in I. \tag{9}$$

Since $\mathsf{v}(f_{G,2})$ must be in $\mathsf{KeyFun}(G, 2, \mathcal{B}, \mathcal{S})$ we get $\mathsf{v}(f_{G,2})(\mathsf{v}(t_1)) = \mathsf{v}(t_2)$. Therefore by the definition of v we have $\mathsf{v}(f_{G,2}(t_1)) = \mathsf{v}(t_2)$. By (9) we obtain $G(\mathsf{v}(t_1), \mathsf{v}(f_{G,2}(t_1))) \in I$ and hence the claim holds. □

Now we are ready to show the reverse direction of Corollary 9.

Theorem 19. *Assume $\mathcal{S} = (\mathcal{R}, \mathcal{K}, \Gamma)$ is a relational schema and $\mathcal{A}$ is an answerset over Γ. Let $\vec{s} \in \Gamma$ and G be a relation symbol of $\mathcal{R}$. If there exist $\vec{t} \in \Gamma_{\mathcal{S}}$ with $G(\vec{t}) \in O_{\mathcal{A}}$ and $\vec{s} \approx \vec{t}$, then $\vec{s} \in \mathsf{cert}(G, \mathcal{S}, \mathcal{A})$.*

Proof. We have to show that for any database instance $\mathcal{B} = (I, \Delta, \approx_{\mathcal{B}})$ in $\mathsf{sem}(\mathcal{S}, \mathcal{A})$ we have $\vec{s} \in G^{\mathcal{B}}$. This can be seen as follows. $G(\vec{t}) \in O_{\mathcal{A}}$ implies that there is a query $Q(\vec{x}) \leftarrow \mathsf{conj}(\vec{x}, \vec{y})$ with $Q(\vec{r}) \in \mathcal{A}$ and $G(\vec{t}) \in Q(\vec{r})^*$. By Lemma 15 there exists a valuation v for $\mathcal{B}$ satisfying $\mathcal{A}$. Hence

$$\mathcal{B}, \mathsf{v} \vdash \mathsf{conj}(\vec{r}, \vec{y}_{Q,\vec{r}}). \tag{10}$$

We additionally have by Lemma 17

$$\mathcal{B}, \mathsf{v} \vdash \vec{t} \approx_{\mathcal{B}} \vec{s}. \tag{11}$$

Applying Lemma 18 to (10) yields $\mathcal{B}, \mathsf{v} \vdash G(\vec{t})$. This is by definition

$$G(\mathsf{v}(\vec{t})) \in I.$$

By (11) we have $\vec{s} = \mathsf{v}(\vec{s}) \approx_{\mathcal{B}} \mathsf{v}(\vec{t})$. Thereby we finally conclude $\vec{s} \in G^{\mathcal{B}}$. □

Finally, we obtain that the canonical database $(O_{\mathcal{A}}, \Gamma_{\mathcal{S}}, \approx)$ contains exactly the certain answers.

Corollary 20. *Assume $\mathcal{S} = (\mathcal{R}, \mathcal{K}, \Gamma)$ is a relational schema and $\mathcal{A}$ is an answerset over Γ. Let $\mathcal{B}$ be the database instance $(O_{\mathcal{A}}, \Gamma_{\mathcal{S}}, \approx)$ and G be a relation symbol of $\mathcal{R}$. For all $\vec{s} \in \Gamma$ we have*

$$\vec{s} \in \mathsf{cert}(G, \mathcal{S}, \mathcal{A}) \text{ if and only if } \vec{s} \in G^{\mathcal{B}}.$$

5 Conclusion

We started this paper with the question of how much information about a database can one infer given access to a view instance and knowledge of the relational schema and its key constraints. We gave an answer to this question by presenting a procedure which computes a minimal database that yields the given view instance and satisfies the relational schema with its key constraints. It is minimal in the sense that it contains exactly the certain answers (with respect to the given view instance) to queries asking about basic predicates.
We only considered views defined by conjunctive queries and we restricted ourselves to unique key constraints. A natural extension of this work would be to allow for more complex view definitions and to consider other forms of database constraints. However, we cannot add arbitrary forms of complexity since the problem may become undecidable, see for example [1, 6].
Regarding our initial question, there is another line of further research. Instead of starting with a view instance, one could begin only with a set of queries over a given schema. Then one could ask whether it is possible with these queries (to which we do *not* know the results) to obtain knowledge about the extension of relations that may not be directly accessed. That is we ask whether there is a set of answers to these queries such that it allows one to deduce information about further predicates. This set of answers can be regarded as a view instance. Hence we are actually asking the question about the existence of such a view instance. This question is also important for security considerations. It allows us to prove that with respect to a given set queries which a user is allowed to perform, it is impossible for him to infer knowledge about relations he cannot directly access.

Acknowledgments. We would like to thank Mathis Kretz for for many helpful comments on an earlier version of this paper.

References

[1] Serge Abiteboul and Oliver M. Duschka. Complexity of answering queries using materialized views. In *Proceedings of PODS'98*, pages 254–265, 1998.

[2] Serge Abiteboul, Richard Hull, and Victor Vianu. *Foundations of Databases*. Addison-Wesley, 1995.

[3] Alfred V. Aho, Yehoshua Sagiv, and Jeffrey D. Ullman. Equivalences among relational expressions. *SIAM Journal on Computing*, 8(2):218–246, 1979.

[4] Michael J. Beeson. *Foundations of Constructive Mathematics: Metamathematical Studies*. Springer, 1985.

[5] Andrea Calì, Diego Calvanese, Giuseppe De Giacomo, and Maurizio Lenzerini. Data integration under integrity constraints. In *Proceedings of the 14th Int. Conference on Advanced Information Systems Engineering CAiSE 2002*, volume 2348 of *Lecture Notes in Computer Science*, pages 262–279. Springer, 2002.

[6] Diego Calvanese and Riccardo Rossi. Answering recursive queries under keys and foreign keys is undecidable. In *Proceedings of the 10th Int. Workshop on Knowledge Representation meets Databases (KRDB 2003)*, pages 3–14. CEUR Electronic Workshop Proceedings, http://ceur-ws.org/Vol-97/, 2003.

[7] Oliver Duschka and Michael R. Genesereth. Answering recursive queries using views. In *Proceedings of the 16th ACM Conference on Principles of Database Systems PODS*, pages 109–116, 1997.

[8] Oliver Duschka and Alon Y. Levy. Recursive plans for information gathering. In *Proceedings of the 15th International Joint Conference on Artificial Intelligence IJCAI*, pages 778–784, 1997.

[9] Oliver M. Duschka, Michael R. Genesereth, and Alon Y. Levy. Recursive query plans for data integration. *The Journal of Logic Programming*, 43:49–74, 2000.

[10] Ronald Fagin, Phokion G. Kolaitis, Renée Miller, and Lucian Popa. Data exchange: Semantics and query answering. To appear in *Theoretical Computer Science*.

[11] Gösta Grahne and Alberto O. Mendelzon. Tableau techniques for querying information sources through global schemas. In C. Beeri and P. Bruneman, editors, *7th Int. Conference on Database Theory (ICDT '99)*, volume 1540 of *Lecture Notes in Computer Science*, pages 332–347. Springer, 1999.

[12] Xiaolei Qian. Query folding. In *Proceedings of the 12th International Conference on Data Engineering*, pages 48–55, 1996.

[13] Jeffrey D. Ullman. *Principles of Database and Knowledge Base Systems*, volume 1. Computer Science Press, 1988.

Addresses

Kilian Stoffel, Université de Neuchâtel,
Pierre-à-Mazel 7, CH-2000 Neuchâtel, Switzerland
kilian.stoffel@unine.ch
Thomas Studer, Institut für Informatik und angewandte Mathematik,
Universität Bern, Neubrückstrasse 10, CH-3012 Bern, Switzerland,
tstuder@iam.unibe.ch