*TLP* **10** (3): 291–329, 2010. © Cambridge University Press 2010
doi:10.1017/S1471068410000104 

# *Querying incomplete data over extended ER schemata*

ANDREA CALÌ
*Computing Laboratory, University of Oxford, Eagle House, Walton Well Road, Oxford OX2 6ED, UK*
(*e-mail:* andrea.cali@comlab.ox.ac.uk)

DAVIDE MARTINENGHI
*Dipartimento di Elettronica e Informazione, Politecnico di Milano*
*Piazza Leonardo 32, 20133 Milano, Italy*
(*e-mail:* davide.martinenghi@polimi.it)

*submitted 24 April 2009; revised 15 November 2009; accepted 1 January 2010*

**Abstract**

Since Chen's Entity-Relationship (ER) model, conceptual modeling has been playing a fundamental role in relational data design. In this paper we consider an extended ER (EER) model enriched with cardinality constraints, disjointness assertions, and is a relations among both entities and relationships. In this setting, we consider the case of incomplete data, which is likely to occur, for instance, when data from different sources are integrated. In such a context, we address the problem of providing correct answers to conjunctive queries by reasoning on the schema. Based on previous results about decidability of the problem, we provide a query answering algorithm that performs rewriting of the initial query into a recursive Datalog query encoding the information about the schema. We finally show extensions to more general settings.

*KEYWORDS*: Extended ER model, dependencies, chase, incomplete data

## 1 Introduction

Conceptual data models, and, in particular, the Entity-Relationship (ER) model (Chen 1976), have long been playing a fundamental role in database design. With the emerging trends in data exchange, information integration, semantic web, and web information systems, the need for dealing with inconsistent and incomplete data has arisen. In this context, it is important to provide correct answers to queries posed over inconsistent and incomplete data (Arenas *et al.* 1999). It is worth noticing here that inconsistency and incompleteness of data is considered with respect to a set of constraints (a.k.a. data dependencies). Such constraints, rather than expressing properties that hold on the data, are used to represent properties of the domain of interest.

We address the problem of answering queries over *incomplete data*, where queries are conjunctive queries expressed over particular relational schemata, called *conceptual schemata*, that are derived from conceptual models. As for the conceptual models, we follow Chen (1976), and we adopt an extension of the well-known ER model, which we call *Extended Entity-Relationship (EER) Model*, along with Thalheim (2000) and the many variants of the classical ER Model. Such an extension is widely adopted in practice and is able to represent classes of objects with their attributes, relationships among classes, cardinality constraints in the participation of entities in relationships, and is a relations among both classes and relationships. We provide a formal semantics to our conceptual model in terms of the relational database model, similarly to what is done in Markowitz and Makowsky (1990). This allows us to formulate conjunctive queries over EER schemata. We do this by providing a translation from EER into relational, whose purpose is to obtain a precise characterization of the relational dependencies that are derived from an EER schema in a design process.

In the presence of data that are incomplete w.r.t. to a set of constraints, we need to *reason* about the dependencies in order to provide certain answers; we do this in a model-theoretic fashion, following the approach of Arenas *et al.* (1999) and Calì *et al.* (2001). Intuitively, we start from a given, incomplete database for the relational schema associated with the EER schema; such data, together with the constraints, are interpreted as a logical theory, with a (possibly infinite) set of models, also called *solutions* in the literature. We adopt the so-called *sound semantics* (see, e.g., Calì *et al.* 2003a): a database is a model if it is a superset of the initial data, and satisfies the constraints. Given a query, the *certain answers* are those that are true in all models.

In this paper we address the problem of answering conjunctive queries over schemata derived from EER schemata in the presence of incomplete data with respect to the schema under the sound semantics. We present an algorithm, based on encoding the information about the conceptual schema and the instance into a *rewriting* of the conjunctive query in Datalog, which computes the certain answers to queries posed in such a context. The algorithm reasons on the integrity constraints and the query.

The problem at hand can be sketchily stated as follows:

- We have a conceptual EER schema. From it, a relational schema $S$ is obtained through a translation mechanism that also produces a set of integrity constraints $\Sigma$ consisting of key and inclusion dependencies.
- We also have an instance $D$ for $S$. $D$ may be inconsistent with respect to $\Sigma$ and incomplete.
- Consider all the $S$-instances that extend $D$ and satisfy $\Sigma$. The certain answers to a conjunctive query $Q$ over $S$ are those that are true of all those instances.
- The problem is how to compute the certain answers to $Q$.
- The solution we propose is to translate $Q$ into a new query $Q^*$ and pose it to $D$. The answers to $Q^*$ are the certain answers to $Q$.

More specifically, our contribution is summarized as follows:

(a) We define a class of relational dependencies, which we call *conceptual dependencies (CDs)* that is able to represent EER schemata; our class is constituted by

https://doi.org/10.1017/S1471068410000104 Published online by Cambridge University Press

a subset of the well-known *key dependencies (KDs)* and *inclusion dependencies (IDs)*. A broad class of KDs and IDs for which the query answering problem under incomplete data is known to be decidable is the class of KDs (at most one per relational predicate) and *nonkey-conflicting inclusion dependences (NKCIDs)*, which was introduced in Calì *et al.* (2003a). The problem of answering incomplete data under general KDs and IDs is known to be undecidable (Calì *et al.* 2003a).

(b) We tackle the problem of query answering under CDs in the presence of incomplete information, under the sound semantics. After reviewing how, also under CDs, the chase is a useful tool for query answering, we solve the problem by means of query rewriting, in the same fashion as in Calì *et al.* (2003b), where a rewriting for KDs and NKCIDs is presented. We show an algorithm that, given a query, rewrites it into another one that encodes relevant information about the relational constraints, so that the evaluation of the rewritten query over the initial incomplete data returns the certain answers. The rewritten query is in (positive) Datalog.

Note that the chase (which we, however, do not construct in our query answering technique) is a conceptual tool whose construction amounts to repairing violations of IDs and KDs, the former by adding tuples, and the latter by merging tuples. However, repairing is not always possible, and in such cases the chase does not exist and query answering becomes trivial. In such cases the repair would require tuple deletions: this is captured by semantics such as those in Bertossi and Bravo (2005) and (Calì *et al.* 2003b).

It is important to notice that the class of CDs does not fall into the class of KDs and NKCIDs. A strong indication (though there is no formal proof) of the decidability, which we show in this paper, of the query answering problem under CDs (and under the sound semantics) is found in Calvanese *et al.* (1998), where it is shown that query containment in a description logic, capable of representing EER schemata, is decidable. However, the technique of Calvanese *et al.* (1998) does not give any indication on the algorithm that may be used to check containment (or, in our case, to answer queries). Differently, our technique gives a direct tool for query answering that, under certain conditions on the data, provides a low computational complexity with respect to the size of the data.

This paper extends the work in Calì (2007) and is organized as follows. We give necessary preliminaries in Section 2; we introduce the EER model in Section 3; in Section 4 we show how to answer queries with the *chase*, a formal tool to deal with dependencies; the query rewriting technique is described in 5, together with extensions to more general cases. Section 6 concludes the paper, discussing related works.

## 2 Preliminaries and notation

In this section we give a formal definition of the relational data model, database constraints, conjunctive queries and answers to queries on incomplete data.

In the relational data model (Codd 1970), predicate symbols are used to denote the relations in the database, whereas constant symbols denote the objects and

https://doi.org/10.1017/S1471068410000104 Published online by Cambridge University Press

the values stored in relations. We assume to have two distinct, fixed, and infinite alphabets $\Gamma_f$ and $\Gamma$ of *fresh constants* and *nonfresh constants* respectively, and we consider only databases over $\Gamma \cup \Gamma_f$. We note that fresh constants are introduced as a technical construct that allows us to build some representatives of databases, as will be explained when introducing the chase. In particular, fresh constants are similar to labeled nulls (Fagin *et al.* 2005) in that they allow representing existentially quantified variables and will thus later be associated with Skolem terms. Indeed, fresh constants play a role analogous to that of Skolem terms. For nonfresh constants, which represent the proper constants of the universe, we adopt the so-called *unique name assumption,* i.e., we assume that different nonfresh constants denote different objects. Instead, fresh constants can be thought of as place holders for nonfresh constants. Therefore, distinct fresh constants can also represent the same object. Furthermore, we shall make use of variables from a set $\Gamma_V$.

A *relational schema* $\mathcal{R}$ consists of an alphabet of *predicate* (or *relation*) symbols, each with an associated *arity* denoting the number of arguments of the predicate (or attributes of the relation). When a relation symbol $r$ has arity $n$, it can be denoted by $r/n$; in general, the arity of $r$ can also be indicated by $arity(r)$.

A *relational database* (or simply database) $D$ over a schema $\mathcal{R}$ is a set of relations with constants as atomic values. We have one relation of arity $n$ for each predicate symbol of arity $n$ in the alphabet $\mathcal{R}$. The relation $r^D$ in $D$ corresponding to the predicate symbol $r$ consists of a set of tuples of constants, which are the tuples satisfying the predicate $r$ in $D$.

When, given a database $D$ for a schema $\mathcal{R}$, a tuple $t = (c_1, \ldots, c_n)$ is in $r^D$, where $r \in \mathcal{R}$, we say that the fact $r(c_1, \ldots, c_n)$ holds in $D$. Henceforth, we will interchangeably use the notion of fact and tuple.

**Integrity constraints.** *Integrity constraints* are assertions on the symbols of the alphabet $\mathcal{R}$ that are intended to be satisfied in every database for the schema. The notion of satisfaction depends on the type of constraints defined over the schema.

The database constraints of interest are IDs and KDs (see, e.g., Abiteboul *et al.* 1995). We denote with over-lined uppercase letters (e.g., $\bar{X}$) both sequences and sets of attributes of relations, and enclose them between vertical bars to denote the number of attributes in the set or sequence (e.g., $|\bar{X}|$). Given a tuple $t$ in relation $r^D$, i.e., a fact $r(t)$ in a database $D$ for a schema $\mathcal{R}$, and a sequence of attributes $\bar{X}$ of $r$, we denote with $t[\bar{X}]$ the *projection* (see e.g. Abiteboul *et al.* 1995) of $t$ on the attributes in $\bar{X}$.

(i) *Inclusion dependencies (IDs).* An ID $\sigma_I$ between relational predicates $r_1$ and $r_2$ is denoted by $r_1[\bar{X}] \subseteq r_2[\bar{Y}]$. Given a database $D$ with values only in $\Gamma$, such a constraint is satisfied in $D$, written $D \models \sigma_I$, iff, for each tuple $t_1$ in $r_1^D$, there exists a tuple $t_2$ in $r_2^D$ such that $t_1[\bar{X}] = t_2[\bar{Y}]$. An ID is said to be a *full-width* ID if every attribute of $r_1$ occurs in $\bar{X}$ exactly once and every attribute of $r_2$ occurs in $\bar{Y}$ exactly once.

(ii) *Key dependencies (KDs).* A KD $\sigma_K$ over a relational predicate $r$ with $arity(r) \geqslant 2$ is denoted by $key(r) = \bar{K}$, where $\bar{K}$ is a nonempty subset of the attributes of $r$. Given a database $D$ with values only in $\Gamma$, such a constraint is satisfied in $D$, written $D \models \sigma_K$, iff, for each $t_1, t_2 \in r^D$ such that $t_1 \neq t_2$, we have $t_1[\bar{K}^*] \neq t_2[\bar{K}^*]$, where $\bar{K}^*$ is any sequence of $|\bar{K}|$ attributes where each attribute in $\bar{K}$ occurs exactly once. Observe that KDs are a special case of functional dependencies (FDs) (Abiteboul *et al.* 1995). Note also that we restricted our definition to predicates with arity at least 2, since for predicates of smaller arity keys would be always satisfied (under set semantics).

Above, we specified when dependencies are satisfied in databases with values only in $\Gamma$. For databases with values in $\Gamma \cup \Gamma_f$, we define satisfaction of dependencies as follows. Given a (key or inclusion) dependency $\sigma$ and a database $D$ with values in $\Gamma \cup \Gamma_f$, let $B$ be a database obtained from $D$ by replacing every distinct fresh constant with a distinct nonfresh constant that does not appear elsewhere in $D$. We have that $\sigma$ is satisfied in $D$, written $D \models \sigma$, iff $B \models \sigma$.

A database $D$ over a schema $\mathscr{R}$ is said to *satisfy* a set of integrity constraints $\Sigma$ expressed over $\mathscr{R}$, written $D \models \Sigma$, if every constraint in $\Sigma$ is satisfied by $D$.

We now briefly introduce the basics of logic programming and Datalog and refer to Lloyd (1987) for further details.

**Logic programs.** Logic programs are formulated in a language $\mathscr{L}$ of predicates and functions of nonnegative arity; 0-ary functions are constants. A language $\mathscr{L}$ is function-free if it contains no functions of arity greater than 0. A *term* is inductively defined as follows: each variable $X$ and each constant $c$ is a term, and if $f$ is an $n$-ary function symbol and $t_1, \ldots, t_n$ are terms, then $f(t_1, \ldots, t_n)$ is a term. A term is *ground* if no variable occurs in it. The *Herbrand universe* of $\mathscr{L}$, denoted $U_{\mathscr{L}}$, is the set of all ground terms that can be formed with the functions and constants in $\mathscr{L}$. An *atom* is a formula $p(t_1, \ldots, t_n)$, where $p$ is a predicate symbol of arity $n$ and each $t_i$ is a term; the atom is ground if all $t_i$ are ground. The *Herbrand base* of a language $\mathscr{L}$, denoted $B_{\mathscr{L}}$, is the set of all ground atoms that can be formed with predicates from $\mathscr{L}$ and terms from $U_{\mathscr{L}}$. A *definite clause* is a rule of the form

$$\underline{A}_0 \leftarrow \underline{A}_1, \ldots, \underline{A}_m \quad (m \geqslant 0),$$

where each $\underline{A}_i$ is an atom. The parts on the left and on the right of "←" are called the *head* and the *body* of the rule, respectively. For a rule $\rho$, we also denote its head by $head(\rho)$, and its body by $body(\rho)$. A rule whose body is empty ($m = 0$) and whose head is ground is called a *fact*. A *logic program* is a set of definite clauses. A clause or logic program is ground if it contains no variables. A clause is *range-restricted* if every variable in it also occurs in its body. A program is range-restricted if all its clauses are.

Each logic program $\Pi$ is associated with the language $\mathscr{L}(\Pi)$ consisting of the predicates, functions, and constants occurring in $\Pi$. If no constant occurs in $\Pi$, we add some constant to $\mathscr{L}(\Pi)$ to have a nonempty domain. We simply write $U_\Pi$ and $B_\Pi$ for $U_{\mathscr{L}(\Pi)}$ and $B_{\mathscr{L}(\Pi)}$, respectively. A *Herbrand interpretation* of a logic program $\Pi$ is any subset $I \subseteq B_\Pi$ of its Herbrand base. Intuitively, the atoms in $I$ are true, and all others are false. A *Herbrand model* of $\Pi$ is a Herbrand interpretation of $\Pi$ such that for each rule $\underline{A}_0 \leftarrow \underline{A}_1, \ldots, \underline{A}_m$ in $\Pi$, this interpretation satisfies the formula $\forall X_1 \ldots \forall X_n (\underline{A}_1 \wedge \cdots \wedge \underline{A}_m) \rightarrow \underline{A}_0$, where $X_1, \ldots, X_n$ are all the variables in the rule.

Let $\Pi$ be a logic program; the *immediate consequence operator* $T_\Pi$ on $\Pi$ is a function from the set of all Herbrand interpretations of $\Pi$ into itself, defined as

$$T_\Pi(I) = \{\underline{A}_0 \in B_\Pi \mid \text{there is } (\underline{A}_0 \leftarrow \underline{A}_1, \ldots, \underline{A}_m) \text{ in } \Pi \text{ and } \{\underline{A}_1, \ldots, \underline{A}_m\} \subseteq I\}.$$

The sequence $T_\Pi^0 = \emptyset, T_\Pi^{i+1} = T_\Pi(T_\Pi^i), i \geqslant 0$ always admits a limit, denoted by $T_\Pi^\infty$, which coincides with the *least Herbrand model* of $\Pi$, i.e., the unique minimal model of $\Pi$ (a model being minimal if no proper subset thereof is also a model). For a set of (ground or nonground) clauses $\Pi$, the immediate consequence operator is defined as $T_\Pi = T_{gr(\Pi)}$, where $gr(\Pi)$ is the set of all clauses obtained from any clause in $\Pi$ by substituting elements of $U_\Pi$ for the variables. A ground atom $\underline{A}$ is called a *consequence* of a set $\Pi$ of clauses if $\underline{A} \in T_\Pi^\infty$, and we write $\Pi \models \underline{A}$.

An $n$-ary query $\Pi_q$ over a schema $\mathscr{R}$ consists of an $n$-ary predicate $q$ (called query predicate) and a finite set $\Pi$ of definite clauses such that

(1) $q$ is the head predicate for at least one rule in $\Pi$;
(2) the predicate symbols of the head atoms are not relation symbols in $\mathscr{R}$;
(3) the predicate symbols of the body atoms are either relation symbols in $\mathscr{R}$ or one of the head predicates of a rule in $\Pi$.

The evaluation, called *answer*, of a query $\Pi_q$ over a database $D$ (which is a set of facts), written $\Pi_q(D)$, is the restriction to $q$ over the least Herbrand model $M$ of the logic program $\Pi \cup D$, i.e., the largest subset of $M$ containing only atoms with predicate $q$. It will be made clear by the context whether by $\Pi_q(D)$ we refer to the set of facts or to the set of tuples in the answer.

A *Datalog* clause is a range-restricted definite clause whose terms are either variables or constants (no function symbols). A Datalog program is a set of Datalog clauses. The notion of query given above also applies to Datalog, since Datalog programs are a specialization of logic programs.

**Conjunctive queries.** In general, a *relational query* is a formula that specifies a set of data to be retrieved from a database. In the following we will refer to the class of conjunctive queries. A *conjunctive query* (CQ) of arity $n$ over a schema $\mathscr{R}$ is a Datalog query $\Pi_q$ such that $\Pi$ consists of a single rule in which

(1) the head is of the form $q(\bar{X})$, where $\bar{X}$ is a sequence of distinct variables;
(2) the constants occurring in the body are from $\Gamma$;
(3) the predicate symbols of the atoms in the body are in $\mathscr{R}$ ($q$ does not occur in the body).

The variables occurring in the head of a conjunctive query are called *distinguished variables*, the others variables occurring in the body are the *nondistinguished variables*. For simplicity, the answer to a conjunctive query $q$ over a database $D$ for $\mathscr{R}$ is more compactly denoted as $q(D)$ (rather than $\Pi_q(D)$).

The answers we are mainly interested in are those that contain no fresh constants, because fresh constants merely represent existentially qualintied variables, in the same way as Skolem terms and labeled nulls (Fagin *et al.* 2005). Therefore we introduce the notation $q^{[\Gamma]}(D)$ for a CQ $q$ to indicate the largest subset of $q(D)$ whose tuples contain no fresh constants.

https://doi.org/10.1017/S1471068410000104 Published online by Cambridge University Press

**Homomorphism.** A *mapping* from one set of symbols, $S_1$, to another set of symbols, $S_2$, is a function $\mu : S_1 \rightarrow S_2$ defined as follows: (i) $\emptyset$ (empty mapping) is a mapping; (ii) if $\mu_0$ is a mapping, then $\mu_0 \cup \{X \rightarrow Y\}$, where $X \in S_1$ and $Y \in S_2$ is a mapping if $\mu_0$ does not already contain some $X \rightarrow Y'$ with $Y \neq Y'$. If $X \rightarrow Y$ is in a mapping $\mu$, we write $\mu(X) = Y$. A *homomorphism* from a set of atoms $D_1$ to another set of atoms $D_2$, both over the same relational schema $\mathscr{R}$, is a mapping $\mu$ from $\Gamma \cup \Gamma_f \cup \Gamma_V$ to $\Gamma \cup \Gamma_f \cup \Gamma_V$ such that the following conditions hold: (1) if $c \in \Gamma$ then $\mu(c) = c$; (2) if $c \in \Gamma_f$ then $\mu(c) \in \Gamma \cup \Gamma_f$; (3) if the atom $r(c_1, \ldots, c_n)$ is in $D_1$, then the atom $r(\mu(c_1), \ldots, \mu(c_n))$ is in $D_2$. In the following, sometimes a homomorphism may have a codomain different from $\Gamma \cup \Gamma_f \cup \Gamma_V$; for instance, it could contain terms from the Herbrand universe of a logic program: in such cases, this will be made explicit.

The notion of homomorphism is naturally extended to atoms as follows. If $\underline{F} = r(c_1, \ldots, c_n)$ is an atom and $\mu$ a homomorphism, we define $\mu(\underline{F}) = r(\mu(c_1), \ldots, \mu(c_n))$. For a *set* of atoms, $F = \{\underline{F}_1, \ldots, \underline{F}_m\}$, we define $\mu(F) = \{\mu(\underline{F}_1), \ldots, \mu(\underline{F}_m)\}$. The set of atoms $\{\mu(\underline{F_1}, \ldots, \mu(\underline{F}_m)\}$ is also called *image* of $F$ with respect to $\mu$. In this case, we say that $\mu$ *maps* $F$ to $\mu(F)$. For a *conjunction* of atoms $\Phi = \underline{F}_1, \ldots, \underline{F}_n$, we use $\mu(\Phi)$ to denote the set of atoms $\mu(\{\underline{F}_1, \ldots, \underline{F}_n\})$. An *isomorphism* is a bijective homomorphism.

**Querying incomplete data.** In the presence of incomplete data, a natural way of considering the problem of query answering is to adopt the so-called *sound semantics* or *open-world assumption* (Reiter 1978; Lenzerini 2002). In this approach, the data are considered sound but not complete, in the sense that they constitute a piece of correct information, but not necessarily all the relevant information. In such a case, we need to reason in the presence of incomplete information, thus considering a theory (given by the schema and constraints) having multiple models. In our context, under relational constraints, it often happens that the data do not satisfy the constraints, especially in information integration, where heterogeneous data are represented by a single schema. Reasoning with incomplete information allows us to address those constraint violations that are caused by the absence of elements from the database (such as IDs). (Note that violations of other kinds of constraints, such as KDs, cannot be addressed in this way.) More formally, we restrict our attention to the so-called *certain answers* to a query: given a finite database $D$, the answers we consider are those that are true in all models, i.e., in *all* the databases that contain $D$ *and* satisfy the dependencies. In the following, we shall always assume that the initial database has finite size, while no finiteness assumptions is made on the models.

*Definition 1* (*Certain answer*)
Consider a relational schema $\mathscr{R}$ with a set of dependencies $\Sigma$, and a finite database $D$ for $\mathscr{R}$. Let $q$ be a conjunctive query of arity $n$ over $\mathscr{R}$. A $n$-tuple $t$ is a *certain answer* to $q$ w.r.t. $D$ and $\Sigma$ if and only if, for every database $B$ for $\mathscr{R}$ such that $B \models \Sigma$ and $B \supseteq D$, we have $t \in q(B)$, and $t$ consists of constants in $\Gamma$. The set of certain answers is denoted by $ans(q, \Sigma, D)$.

*Example 1*
Consider a relational schema $\mathscr{R}$, here inspired by Calì *et al.* (2003b), with the relations $\mathsf{player}/2$ (player-team pairs) and $\mathsf{team}/2$ (team-city pairs), a set of IDs $\Sigma = \{\mathsf{player}[2] \subseteq \mathsf{team}[1]\}$, and a database $D$ consisting of the facts $\mathsf{player}(\mathit{pirlo}, \mathit{acMilan})$, $\mathsf{player}(\mathit{totti}, \mathit{roma})$, $\mathsf{team}(\mathit{acMilan}, \mathit{milan})$.

The ID in $\Sigma$ tells us that *roma* is the name of some team in every database $B \supseteq D$ such that $B \models \Sigma$, i.e., each such database $B$ must contain at least a fact of the form $\mathsf{team}(\mathit{roma}, c)$, where $c$ is some value in $\Gamma$.

Consider now the query $q(X) \leftarrow \mathsf{team}(X, Y)$, asking the names of the teams in the database. By the above considerations, the set of certain answers is $\{\mathit{acMilan}, \mathit{roma}\}$.

Let $\underline{F}$ be the fact $\mathsf{team}(\mathit{roma}, \alpha)$, where $\alpha$ is a value in $\Gamma_f$. As we will show in Section 4, there is a homomorphism from $D \cup \{\underline{F}\}$ to every database $B' \supset D$ such that $B' \models \Sigma$. Consider, e.g., such a database $B' = \{\mathsf{player}(\mathit{pirlo}, \mathit{acMilan}), \mathsf{player}(\mathit{totti}, \mathit{roma}), \mathsf{team}(\mathit{acMilan}, \mathit{milan}), \mathsf{team}(\mathit{roma}, \mathit{rome}), \mathsf{team}(\mathit{psg}, \mathit{paris})\}$. There is a homomorphism $\lambda$ from $D \cup \{\underline{F}\}$ to $B'$ such that (i) $\lambda(\alpha) = \mathit{rome}$, (ii) $\lambda(\underline{F}) = \mathsf{team}(\mathit{roma}, \mathit{rome})$, (iii) $\lambda$ sends all facts in $D$ into themselves, and (iv) $B' = \lambda(D \cup \{\underline{F}\}) \cup \{\mathsf{team}(\mathit{psg}, \mathit{paris})\}$.

We will see that, under the database dependencies we consider in this paper, the problem of query answering is mainly complicated by two facts: (i) the number of databases that satisfy $\Sigma$ and that include $D$ can be infinite; (ii) there is no bound to the size of such databases.

*Definition 2 (Querying incomplete databases)*
Consider a relational schema $\mathscr{R}$ with a set of dependencies $\Sigma$, and a finite database $D$ for $\mathscr{R}$. Let $q$ be a conjunctive query of arity $n$ over $\mathscr{R}$. The problem of querying incomplete databases under $\Sigma$ is the problem of determining *all* tuples in $\mathit{ans}(q, \Sigma, D)$. The corresponding decision problem is determining, given also a tuple $t$ of arity $n$, whether $t \in \mathit{ans}(q, \Sigma, D)$.

## 3 The conceptual model

In this section we present the conceptual model we shall deal with in the rest of the paper, and we give its semantics in terms of relational database schemata with constraints.

Such model incorporates the basic features of the ER model (Chen 1976) and OO models, including subset (or is-a) constraints on both entities and relationships. It is an extension of the one presented in Calì *et al.* (2001), and here we use a notation analogous to that of Calì *et al.* (2001). Henceforth, we will call such a model *EER model*, and we will call schemata expressed in the EER model *EER schemata.*

An *EER schema* consists of a collection of entity, relationship, and attribute definitions over an *alphabet Sym of symbols.* The alphabet *Sym* is partitioned into a set of entity symbols (denoted by *Ent*), a set of relationship symbols (denoted by *Rel*), and a set of attribute symbols (denoted by *Att*).

An *entity definition* has the form

entity $E$
  isa: $E_1, \ldots, E_h$
  participates($\geqslant 1$): $R_1 : c_1, \ldots, R_\ell : c_\ell$
  participates($\leqslant 1$): $R'_1 : c'_1, \ldots, R'_{\ell'} : c'_{\ell'}$

where: (i) $E \in Ent$ is the entity to be defined; (ii) the isa clause specifies a set of entities to which $E$ is related via is-a (i.e., the set of entities that are supersets of $e$); (iii) the participates($\geqslant 1$) clause specifies those relationships in which an instance of $E$ must necessarily participate; and for each relationship $R_i$, the clause specifies that $E$ participates as $c_i$-th component in $R_i$; (iv) the participates($\leqslant 1$) clause specifies those relationships in which an instance of $E$ cannot participate more than once (components are specified as in the previous case). The isa, participates($\geqslant 1$) and participates($\leqslant 1$) clauses are optional. Every relationship mentioned in the participates($\leqslant 1$) and participates($\geqslant 1$) clauses must then be defined accordingly, by mentioning the participating entity as one of the entities of the relationship in a relationship definition. A *relationship definition* has the form

relationship $R$ among $E_1, \ldots, E_n$
  isa: $R_1[j_{1\,1}, \ldots, j_{1\,n}], \ldots, R_h[j_{h\,1}, \ldots, j_{h\,n}]$

where: (i) $R \in Rel$ is the relationship to be defined; (ii) the $n$ entities of $Ent$, with $n \geqslant 2$, listed in the among clause are those among which the relationship is defined (i.e., component $i$ of $R$ is an instance of entity $E_i$); (iii) the isa clause specifies a set of relationships to which $R$ is related via is-a; for each relation $R_i$, we specify in square brackets how the components $[1, \ldots, n]$ are related to those of $e_i$, by specifying a permutation $[j_{i\,1}, \ldots, j_{i\,n}]$ of the components of $E_i$; (iv) the number $n$ of entities in the among clause is the *arity* of $R$. The isa, clause is optional. An *attribute definition* has the form

attribute $A$ of $X$
  *qualification*

where: (i) $A \in Att$ is the attribute to be defined; (ii) $X$ is the entity or relationship with which the attribute is associated; (iii) *qualification* consists of none, one, or both of the keywords functional and mandatory, specifying respectively that each instance of $X$ has a unique value for attribute $A$, and that each instance of $X$ needs to have at least a value for attribute $A$. If the functional or mandatory keywords are missing, the attribute is assumed by default to be multivalued and optional, respectively.

For the sake of simplicity, and without any loss of generality, we assume that in our EER model attributes of entities or relationships have unique names in a schema. We also assume that every attribute or entity takes values from an infinite domain.

The semantics of an EER schema $\mathscr{C}$ is defined by (i) associating a relational schema $\mathscr{R}$ to it, and (ii) specifying when a database for $\mathscr{R}$ satisfies all constraints imposed by the constructs of the schema $\mathscr{C}$.

https://doi.org/10.1017/S1471068410000104 Published online by Cambridge University Press

We now formally define the relational schema associated with an EER diagram. Such a relational schema is defined in terms of *predicates*, which represent the so-called concepts (entities, relationships, and attributes) of the EER schema.

(a) Each entity $E$ in $\mathscr{C}$ has an associated predicate $e$ of arity 1. Informally, a fact of the form $e(c)$ asserts that $c$ is an instance of entity $E$.
(b) Each attribute $A$ for an entity $E$ in $\mathscr{C}$ has an associated predicate $a$ of arity 2. Informally, a fact of the form $a(c, d)$ asserts that $d$ is the value of attribute $A$ associated with $c$, where $c$ is an instance of entity $E$.
(c) Each relationship $R$ involving the entities $E_1, \ldots, E_n$ in $\mathscr{C}$ has an associated predicate $r$ of arity $n$. Informally, a fact of the form $r(c_1, \ldots, c_n)$ asserts that $(c_1, \ldots, c_n)$ is an instance of relationship $R$, where $c_1, \ldots, c_n$ are instances of $E_1, \ldots, E_n$ respectively.
(d) Each attribute $A$ for a relationship $R$ among the entities $E_1, \ldots, E_n$ in $\mathscr{C}$ has an associated predicate $a$ of arity $n+1$. Informally, a fact of the form $a(c_1, \ldots, c_n, d)$ asserts that $d$ is a value of attribute $A$ associated with the instance $(c_1, \ldots, c_n)$ of relationship $R$.

Notice that, in our particular relational representation, entities are represented by unary predicates, which can be thus seen as "surrogate keys," i.e., attributes that are identifiers and do not have any real-world meaning. With this representation, user-defined key attributes are not necessary.

In the following, the expression "query over an EER schema $\mathscr{C}$" will indicate a query over the relational schema associated wih $\mathscr{C}$ according to the above points (a) to (d).

*Example 2*
Consider the EER schema $\mathscr{C}$ defined as follows.

```
entity Employee
  participates(⩾ 1): Works_in : 1
  participates(⩽ 1): Works_in : 1
entity Manager
  isa: Employee
  participates(⩾ 1): Manages : 1
  participates(⩽ 1): Manages : 1
entity Dept
relationship Works_in among Employee, Dept
relationship Manages among Manager, Dept
  isa: Works_in[1, 2]
attribute emp_name of Employee
attribute dept_name of Dept
attribute since of Works_in
```

Figure 1 depicts $\mathscr{C}$ in the usual graphical notation for the ER model (components are indicated by integers for the relationships). The relational schema $\mathscr{R}$ associated with $\mathscr{C}$ consists of the predicates manager/1, employee/1, dept/1, works_in/2, manages/2, emp_name/2, dept_name/2, since/3. The schema describes employees

https://doi.org/10.1017/S1471068410000104 Published online by Cambridge University Press

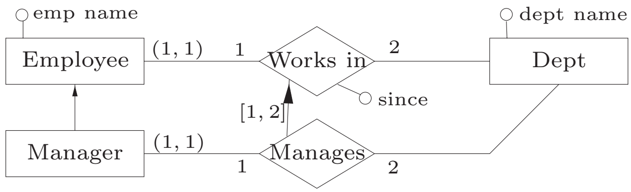


Fig. 1. EER schema for Example 2.

working in departments of a firm, and managers that are also employees, and manage departments. Managers who manage a department also work in the same department, as imposed by the is-a among the two relationships; the permutation [1, 2] labeling the arrow denotes that the is-a holds considering the components in the same order (in general, any permutation of $(1, \ldots, n)$ is possible for an is-a between two $n$-ary relationships). The constraint (1, 1) on the participation of Employee in Works_In imposes that every instance of Employee participates at least once (mandatory participation) and at most once (functional participation) in Works_In; the same constraints hold on the participation of Manager in Manages. Suppose we want to know the names of the managers who manage the toy department (named *toy_dept*). The corresponding conjunctive query over $\mathscr{C}$ is

$$q(Z) \quad \leftarrow \quad \mathsf{manager}(X), \mathsf{emp\_name}(X, Z), \mathsf{manages}(X, Y), \mathsf{dept}(Y), \\ \mathsf{dept\_name}(Y, toy\_dept)$$

The intended semantics of an EER schema is immediately captured by a translation into the relational model that imposes additional constraints to the associated relational schema. Once we have defined the relational schema $\mathscr{R}$ for an EER schema $\mathscr{C}$, we give the semantics of each construct of the EER model; this is done by specifying what databases (i.e., extensions of the predicates of $\mathscr{R}$) *satisfy* the constraints imposed by the constructs of the EER diagram. We do that by making use of the relational database constraints introduced in Section 2. We remind the reader that each entity $E$ in $\mathscr{C}$ has an associated relational predicate $e$ in $\mathscr{R}$, denoted with the same letter, lowercase instead of uppercase; similarly, an attribute $A$ has associated a predicate $a$ and a relationship $R$ a predicate $r$.

(1) For each attribute $A/2$ for an entity $E$ in an attribute definition in $\mathscr{C}$, we have the ID $a[1] \subseteq e[1]$.
(2) For each attribute $A/(n+1)$ for a relationship $R/n$ in an attribute definition in $\mathscr{C}$, we have the ID $a[1, \ldots, n] \subseteq r[1, \ldots, n]$.
(3) For each relationship $R$ involving an entity $E_i$ as i-th component according to the corresponding relationship definition in $\mathscr{C}$, we have the ID $r[i] \subseteq e_i[1]$.
(4) For each mandatory attribute $A/2$ of an entity $E$ in an attribute definition in $\mathscr{C}$, we have the ID $e[1] \subseteq a[1]$.
(5) For each mandatory attribute $A/(n+1)$ of a relationship $R/n$ in an attribute definition in $\mathscr{C}$, we have the ID $r[1, \ldots, n] \subseteq a[1, \ldots, n]$.
(6) For each functional attribute $A/2$ of an entity $E$ in an attribute definition in $\mathscr{C}$, we have the KD $key(a) = \{1\}$, since there cannot be more than one value for attribute $A$ that is assigned to a single instance of $E$.

https://doi.org/10.1017/S1471068410000104 Published online by Cambridge University Press

(7) For each functional attribute $A/(n+1)$ of a relationship $R/n$ in an attribute definition of $\mathscr{C}$, we have the KD $key(a) = \{1, \ldots, n\}$, since there cannot be more than one value for attribute $A$ that is assigned to a single instance of $R$.
(8) For each is-a relation between entities $E_1$ and $E_2$, in an entity definition in $\mathscr{C}$, we have the ID $e_1[1] \subseteq e_2[1]$, since the is-a relation specifies a set containment between entities $E_1$ and $E_2$.
(9) For each is-a relation between relationships $R_1$ and $R_2$, where components $1, \ldots, n$ of $R_1$ correspond to components $j_1, \ldots, j_n$, in a relationship definition in $\mathscr{C}$, we have the ID: $r_1[1, \ldots, n] \subseteq r_2[j_1, \ldots, j_n]$, since the is-a relation specifies a set containment between relationships $R_1$ and $R_2$.
(10) For each mandatory participation (participation with minimum cardinality 1) as $c$-th component of an entity $E$ in a relationship $R$, specified by a clause participates$\geqslant 1$: $R : c$ in an entity definition in $\mathscr{C}$, we have the ID $e[1] \subseteq r[c]$.
(11) For each participation with maximum cardinality 1 as $c$-th component of an entity $E$ in a relationship $R$, specified by a clause participates$\leqslant 1$: $R : c$ in an entity definition in $\mathscr{C}$, we have the KD $key(r) = \{c\}$.

*Definition 3* (*Conceptual dependencies*)
Consider a schema $\mathscr{R}$ and a set of dependencies $\Sigma = \Sigma_I \cup \Sigma_K$, where $\Sigma_I$ is a set of IDs and $\Sigma_K$ is a set of KDs expressed over $\mathscr{R}$. We say that $\Sigma$ is a set of CDs if there exists an EER schema $\mathscr{C}$ with associated relational schema $\mathscr{R}$ such that $\Sigma$ is obtained from $\mathscr{C}$ by applying the above points (1)–(11).

*Example 2* (*cont.*)
Consider again the EER schema shown in Figure 1. The set of CDs associated with the EER schema $\mathscr{C}$ to be imposed on the schema $\mathscr{R}$ consists of the following dependencies.

$$
\begin{array}{rrcll}
\sigma_1 : & \mathsf{dept\_name}[1] & \subseteq & \mathsf{dept}[1] & \text{(by rule 1)} \\
\sigma_2 : & \mathsf{emp\_name}[1] & \subseteq & \mathsf{employee}[1] & \text{(by rule 1)} \\
\sigma_3 : & \mathsf{since}[1,2] & \subseteq & \mathsf{works\_in}[1,2] & \text{(by rule 2)} \\
\sigma_4 : & \mathsf{works\_in}[1] & \subseteq & \mathsf{employee}[1] & \text{(by rule 3)} \\
\sigma_5 : & \mathsf{works\_in}[2] & \subseteq & \mathsf{dept}[1] & \text{(by rule 3)} \\
\sigma_6 : & \mathsf{manages}[1] & \subseteq & \mathsf{manager}[1] & \text{(by rule 3)} \\
\sigma_7 : & \mathsf{manages}[2] & \subseteq & \mathsf{dept}[1] & \text{(by rule 3)} \\
\sigma_8 : & \mathsf{manager}[1] & \subseteq & \mathsf{employee}[1] & \text{(by rule 8)} \\
\sigma_9 : & \mathsf{manages}[1,2] & \subseteq & \mathsf{works\_in}[1,2] & \text{(by rule 9)} \\
\sigma_{10} : & \mathsf{employee}[1] & \subseteq & \mathsf{works\_in}[1] & \text{(by rule 10)} \\
\sigma_{11} : & \mathsf{manager}[1] & \subseteq & \mathsf{manages}[1] & \text{(by rule 10)} \\
\sigma_{12} : & key(\mathsf{works\_in}) & = & \{1\} & \text{(by rule 11)} \\
\sigma_{13} : & key(\mathsf{manages}) & = & \{1\} & \text{(by rule 11)}
\end{array}
$$

Now we characterize the form of relational dependencies resulting from the encoding of EER schemata into relational schemata, the proof of which is straightforward.

*Proposition 1*
Consider a schema $\mathscr{R}$ and a set of dependencies $\Sigma = \Sigma_I \cup \Sigma_K$, where $\Sigma_I$ is a set of IDs and $\Sigma_K$ is a set of KDs expressed over $\mathscr{R}$. Then, $\Sigma$ is a set of CDs if and only if we can partition $\mathscr{R}$ in three sets $\mathscr{R}_R$, $\mathscr{R}_E$, and $\mathscr{R}_A$ such that the following holds.

(a) All predicate symbols in $\mathscr{R}_E$ are unary.
(b) All predicate symbols in $\mathscr{R}_R$ and $\mathscr{R}_A$ have arity at least 2.
(c) The dependencies in $\Sigma_K$ have one of the following forms
   (1) $key(r) = \{i\}$, with $1 \leqslant i \leqslant arity(r)$, where $r \in \mathscr{R}_R$.
   (2) $key(a) = \{1, \ldots, n\}$, where $a \in \mathscr{R}_A$ and $n = arity(a) - 1$.
(d) The dependencies in $\Sigma_I$ have one of the following forms
   (1) $e_1[1] \subseteq e_2[1]$, where $\{e_1, e_2\} \subseteq \mathscr{R}_E$.
   (2) $e[1] \subseteq r[i]$, where $e \in \mathscr{R}_E$, $r \in \mathscr{R}_R$, and $1 \leqslant i \leqslant arity(r)$.
   (3) $r[i] \subseteq e[1]$, where $r \in \mathscr{R}_R$, $e \in \mathscr{R}_E$, and $1 \leqslant i \leqslant arity(r)$.
   (4) $r_1[1, \ldots, k] \subseteq r_2[i_1, \ldots, i_k]$, where $\{r_1, r_2\} \subseteq \mathscr{R}_R$, $arity(r_1) = arity(r_2) = k$, and $(i_1, \ldots, i_k)$ is a permutation of $(1, \ldots, k)$.
   (5) $a[1] \subseteq e[1]$, where $a \in \mathscr{R}_A$ and $e \in \mathscr{R}_E$.
   (6) $a[1, \ldots, n] \subseteq r[1, \ldots, n]$, where $a \in \mathscr{R}_A$, $r \in \mathscr{R}_R$, and $n = arity(r) = arity(a) - 1$.
   (7) $e[1] \subseteq a[1]$, where $e \in \mathscr{R}_E$ and $a \in \mathscr{R}_A$.
   (8) $r[1, \ldots, n] \subseteq a[1, \ldots, n]$, where $r \in \mathscr{R}_R$, $a \in \mathscr{R}_A$, and $n = arity(r) = arity(a) - 1$.
(e) For every predicate $r \in \mathscr{R}_R$ and for $1 \leqslant i \leqslant arity(r)$, there exists an ID $r[i] \subseteq e_i[1]$ in $\Sigma_I$ such that $e_i \in \mathscr{R}_E$ and there is no $e_i' \in \mathscr{R}_E$, with $e_i \neq e_i'$, such that $r[i] \subseteq e_i'[1]$ is in $\Sigma_I$.
(f) For every predicate $a \in \mathscr{R}_A$, there exists an ID $a[1, \ldots, n] \subseteq p[1, \ldots, n]$ in $\Sigma_I$ such that $p \in \mathscr{R}_R \cup \mathscr{R}_E$ and $n = arity(p) = arity(a) - 1$, and there is no $p' \in \mathscr{R}_R \cup \mathscr{R}_E$, with $p \neq p'$, such that $a[1, \ldots, n] \subseteq p'[1, \ldots, n]$ is in $\Sigma_I$.
(g) For every ID $e[1] \subseteq r[i]$ in $\Sigma_I$, with $e \in \mathscr{R}_E$, $r \in \mathscr{R}_R$, and $1 \leqslant i \leqslant arity(r)$, there is an ID $r[i] \subseteq e[1]$ in $\Sigma_I$.
(h) For every ID $r[1, \ldots, n] \subseteq a[1, \ldots, n]$ in $\Sigma_I$, with $r \in \mathscr{R}_R$, $a \in \mathscr{R}_A$, and $n = arity(r) = arity(a) - 1$, there is an ID $a[1, \ldots, n] \subseteq r[1, \ldots, n]$ in $\Sigma_I$.
(i) For every ID $e[1] \subseteq a[1]$ in $\Sigma_I$, with $e \in \mathscr{R}_E$, $a \in \mathscr{R}_A$, and $arity(a) = 2$, there is an ID $a[1] \subseteq e[1]$ in $\Sigma_I$.

Being able to encode EER schemata into relational ones, henceforth we will deal with relational schemata only.

The problem of querying incomplete databases under KDs and IDs is in general undecidable (Calì 2003; Calì *et al.* 2003a). The largest subclass of functional dependencies[1] and IDs for which query answering is known to be decidable is the class of keys and nonkey conflicting inclusion dependencies (Calì 2003; Calì *et al.* 2003a). The main contribution of the present paper is a technique for solving the problem of querying incomplete databases under CDs. This is relevant because

[1] Functional dependencies are a generalization of KDs (Abiteboul *et al.* 1995).

https://doi.org/10.1017/S1471068410000104 Published online by Cambridge University Press

EER schemata are very important in practice and CDs are able to capture them. Our solution consists in a technique for rewriting the given query such that the evaluation of the rewritten query returns the certain answers.

Note that our definition of certain answer, defined in Section 2, considers databases that may also be of infinite size. In the database literature, interest is typically devoted to databases of finite size only. In particular, the certain answers under finite models can be defined as follows.

*Definition 4* (*Certain answer under finite models*)
Consider a relational schema $\mathscr{R}$ with a set of dependencies $\Sigma$, and a finite database $D$ for $\mathscr{R}$. Let $q$ be a conjunctive query of arity $n$ over $\mathscr{R}$. A $n$-tuple $t$ is a *certain answer under finite models* to $q$ w.r.t. $D$ and $\Sigma$ if and only if, for every *finite* database $B$ for $\mathscr{R}$ such that $B \models \Sigma$ and $B \supseteq D$, we have $t \in q(B)$, and $t$ consists of constants in $\Gamma$. The set of certain answers under finite models is denoted by $ans_f(q, \Sigma, D)$.

We now show that under CDs, in general, $ans(q, \Sigma, D) \neq ans_f(q, \Sigma, D)$.

*Example 3*
Consider the following EER schema:

entity B
  participates($\geqslant 1$): R : 2
entity A
  isa: B
  participates($\leqslant 1$): R : 1
relationship R among A, B

This corresponds to the following set of CDs:

$$\Sigma = \begin{cases} r[1] \subseteq a[1], \\ r[2] \subseteq b[1], \\ a[1] \subseteq b[1], \\ b[1] \subseteq r[2], \\ key(r) = \{1\}. \end{cases}$$

It can be straightforwardly seen that, for every *finite* database $B \supseteq D$ such that $B \models \Sigma$, we have $a(c) \in B$. Consequently, $\langle c \rangle \in ans_f(q, \Sigma, D)$, where $q$ is the query $q(x) \leftarrow a(x)$. On the other hand, consider the following database $D_\infty$.

$$\begin{aligned} D_\infty = \{\, & b(c), r(c_1, c), a(c_1), b(c_1), r(c_2, c_1), a(c_2), b(c_2), r(c_3, c_2), \ldots \\ & \ldots, a(c_i), b(c_i), r(c_{i+1}, c_i), \ldots \}. \end{aligned}$$

We have that $D_\infty \supseteq D$ and $D_\infty \models \Sigma$, but $a(c) \notin D_\infty$ and thus $\langle c \rangle \notin ans(q, \Sigma, D)$, therefore we immediately have $ans(q, \Sigma, D) \neq ans_f(q, \Sigma, D)$.

Henceforth, we shall not restrict our attention to finite databases only, thus allowing for models of infinite size.

## 4 Query answering with the chase

In this section we introduce the notion of *chase*, which is a fundamental tool for dealing with database constraints (Maier *et al.* 1979; Maier *et al.* 1981; Vardi 1983;

Johnson and Klug 1984); then we show some relevant properties of the chase under CDs regarding conjunctive query answering, which will pave the way for the query rewriting technique that will be presented in the next section.

The *chase* (Maier *et al.* 1979; Johnson and Klug 1984) is a key concept in particular in the context of functional and IDs. Intuitively, given a database, its facts in general do not satisfy the dependencies; the idea of the chase is to convert the initial facts into a new set of facts constituting a database that satisfies the dependencies, possibly by collapsing facts (according to KDs) or adding new facts (according to IDs). When new facts are added, some of the constants need to be *fresh*, as we shall see in the following. The technique to construct a chase is well known for functional and IDs (see, e.g., Johnson and Klug 1984); however we detail this technique here, since we have adapted it to the simpler case of KDs instead of FDs.

### *4.1 Construction of the chase*

In order to construct the chase for a database for a relational schema $\mathscr{R}$ with dependencies $\Sigma = \Sigma_I \cup \Sigma_K$, where $\Sigma_I$ is a set of IDs and $\Sigma_K$ is a set of KDs, we use the following rules for IDs and KDs, which apply to a set of facts (i.e., a database instance) and produce a new set of facts. We indicate as $D$ the set of facts before the application of a rule.

Inclusion Dependency Chase Rule. Let $r, s$ be relational symbols in $\mathscr{R}$. Suppose there is a tuple $t$ in $r^D$, and there is an ID $\sigma \in \Sigma_I$ of the form $r[\bar{X}_r] \subseteq s[\bar{X}_s]$. If there is no tuple $t'$ in $s^D$ such that $t'[\bar{X}_s] = t[\bar{X}_r]$ (in this case we say the rule is *applicable*), then we add a new tuple $t_{chase}$ in $s^D$ such that $t_{chase}[\bar{X}_s] = t[\bar{X}_r]$, and for every attribute $A_i$ of $s$ such that $A_i \notin \bar{X}_s$, $t_{chase}[A_i]$ is a fresh value in $\Gamma_f$ that *follows*, according to lexicographic order, all the values already present in the chase. Note also that we assume that all the values in $\Gamma_f$ follow, according to lexicographic order, all the values in $\Gamma$.

Key Dependency Chase Rule. Let $r$ be a relational symbol in $\mathscr{R}$. Suppose there is a KD $\kappa$ of the form $key(r) = \bar{X}$. If there are two *distinct* tuples $t, t' \in r^D$ such that $t[\bar{X}] = t'[\bar{X}]$ (in this case we say the rule is *applicable*), make the symbols in $t$ and $t'$ equal in the following way. Let $\bar{Y} = Y_1, \ldots, Y_\ell$ be the attributes of $r$ that are not in $\bar{X}$; for all $i \in \{1, \ldots, \ell\}$, make $t[Y_i]$ and $t'[Y_i]$ merge into a combined symbol according to the following criterion: (i) if both are constants in $\Gamma$ and they are not equal, the rule fails to apply and the chase construction process is halted; (ii) if one is in $\Gamma$ and the other is a fresh constant in $\Gamma_f$, let the combined symbol be the nonfresh constant; (iii) if both are in $\Gamma_f$, let the combined symbol be the one preceding the other in lexicographic order. Finally, replace all occurrences in $D$ of $t[Y_i]$ and $t'[Y_i]$ with their combined symbol.

Now we come to the formal definition of the chase, which uses the notion of *level* of a tuple; intuitively, the lower the level of a tuple, the earlier the tuple has been constructed in the chase. In order to make all steps in the construction of the chase univocally determined by the definition, we assume that all facts can be sorted according to lexicographic order (e.g., by using a string comprising the predicate name and the names of all constants in the fact), and so can all pairs of facts as well as all dependencies (e.g., also by using strings that encode them).

*Definition 5* (*Chase*)

Let $D$ be a database for a schema $\mathscr{R}$, and $\Sigma$ a set of CDs. We call *chase* of $D$ according to $\Sigma$, denoted $chase_\Sigma(D)$, the database constructed from $D$ by repeatedly executing the following steps, while the KD and ID chase rules are applicable; every tuple $t \in chase_\Sigma(D)$ is also assigned a *level*, denoted by $level(t)$; if $t \in D$, then $level(t) = 0$.

(1) While there are pairs of facts on which the KD chase rule is applicable, take the pair $t_1, t_2$ such that $min(level(t_1), level(t_2))$ is minimal (if there is more than one, take the pair that comes first in lexicographic order) and apply the KD chase rule on $t_1, t_2$ w.r.t. a KD $\kappa$ (if there is more than one KD for which the KD chase rule is applicable on $t_1, t_2$, take the KD that comes first in lexicographic order) so that $t_1, t_2$ collapse into a fact $t_3$; if the rule fails, the chase cannot be constructed and, thus, does not exist; else we define $level(t_3) = min(level(t_1), level(t_2))$.

(2) If there are facts on which the ID chase rule is applicable w.r.t. a full-width ID, choose *the one* (say $t'$) at the lowest level that lexicographically comes first and apply the ID chase rule on $t'$ w.r.t. a full-width ID $\sigma$ (if there is more than one full-width ID for which the ID chase rule is applicable on $t'$, take the full-width ID that comes first in lexicographic order) to generate a new fact $t$;″ else, if there are facts on which the ID chase rule is applicable, choose *the one* (say $t'$) at the lowest level that lexicographically comes first and apply the ID chase rule on $t'$ w.r.t. an ID $\sigma$ (if there is more than one ID for which the ID chase rule is applicable on $t'$, take the ID that comes first in lexicographic order) to generate a new fact $t$.″ We define $level(t'') = level(t') + 1$.

Note that, according to Definition 5, the chase is constructed by applying the KD chase rule as long as possible, then the ID chase rule exactly once, then the KD chase rule as long as possible, etc., until no more rule is applicable. Also, the particular sequence of chase rules to be applied is determined according to a precise lexicographic order, so that there is exactly one chase for a given initial database and set of CDs.

As we pointed out before, the aim of the construction of the chase is to make the initial database satisfy the KDs and the IDs, by repairing the violations of the constraints. The obtained (possibly infinite) instance is a representative of all databases that are a superset of the initial database and satisfy the constraints. Notice that KD violations cannot be repaired by constructing a chase, but would require an explicit treatment, as explained in Section 5.4; in such a case the chase does not exist. It is easy to see that $chase_\Sigma(D)$ can be infinite only if the set of IDs in $\Sigma$ is *cyclic* (Johnson and Klug 1984; Abiteboul *et al.* 1995), i.e., if there is a sequence of IDs in $\Sigma$ of the form $r_1[\bar{X}_1] \subseteq r_2[\bar{X}'_1], r_2[\bar{X}_2] \subseteq r_3[\bar{X}'_2], \ldots, r_n[\bar{X}_n] \subseteq r_{n+1}[\bar{X}'_n]$ and $r_{n+1} = r_1$. In the following we will show how the chase can be used in computing the answers to queries over incomplete databases under dependencies.

https://doi.org/10.1017/S1471068410000104 Published online by Cambridge University Press

### *4.2 Query answering and the chase*

In their milestone paper (Johnson and Klug 1984), Johnson and Klug proved that, under certain subclasses of KDs and IDs, a containment between two conjunctive queries $q_1$ and $q_2$ can be tested by verifying the existence of a so-called query homomorphism. Roughly speaking, such a homomorphism has to map the body of $q_2$ to the chase of the body of $q_1$, and the head of $q_2$ to the head of $q_1$. Johnson and Klug proved that, in order to test containment of CQs under IDs alone or *key-based* dependencies (a special class of KDs and IDs), it is sufficient to consider a *finite*, initial portion of the chase. The result of Johnson and Klug (1984) was extended in Calì *et al.* (2003a) to a broader class of dependencies, strictly more general than keys with foreign keys: the class of KDs and *nonkey-conflicting inclusion dependencies (NKCIDs)* (Calì 2003), which behave like IDs alone because NKCIDs do not interfere with KDs in the construction of the chase. The above results about query containment (see, e.g., Calì *et al.* 2008) can be straightforwardly adapted to solve the decision problem of answering on incomplete databases, since, as it will be shown later, the chase is a representative of all databases that satisfy the dependencies and are a superset of the initial data.

In a set of CDs, IDs are not nonkey-conflicting (or better *key-conflicting*), therefore the decidability of query answering cannot be deduced from (Johnson and Klug 1984; Calì *et al.* 2003a), (though it can be derived from (Calvanese *et al.* 1998), as we shall discuss later). In particular, under CDs, the construction of the chase has to face interactions between KDs and IDs; this can be seen in the following example, taken from (Calì 2006).

*Example 4*

Consider again the EER schema of Example 2. Suppose we have an initial (incomplete) database, with the facts $\mathsf{manager}(m)$ and $\mathsf{works\_in}(m,d)$. If we construct the chase, we obtain the facts $\mathsf{employee}(m)$, $\mathsf{manages}(m,\alpha_1)$, $\mathsf{works\_in}(m,\alpha_1)$, $\mathsf{dept}(\alpha_1)$, where $\alpha_1$ is a fresh constant. Observe that $m$ cannot participate more than once in $\mathsf{works\_in}$, so we deduce $\alpha_1 = d$. We must therefore replace $\alpha_1$ with $d$ in the rest of the chase, including the part that has been constructed so far. Therefore, $chase_\Sigma(D) = \{\mathsf{manager}(m), \mathsf{works\_in}(m,d), \mathsf{employee}(m), \mathsf{manages}(m,d), \mathsf{dept}(d)\}$.

In spite of the potentially harmful interaction between IDs and KDs, analogously to the case of IDs alone (Calì *et al.* 2004), it can be proved that, in the presence of CDs, the chase is a representative of all databases that are a superset of the initial (incomplete) data, and satisfy the dependencies; therefore, it serves as a tool for query answering, as shown in Theorem 1 below.

As was made explicit in Definition 5, the chase may not exist if some application of the KD rule fails. This may happen even when the database satisfies the KDs, as shown in the next example.

*Example 5*

Consider two binary predicates $r$ and $s$, derived from two binary relationships $R$ and $S$, for which there is an is-a relation (that generates the ID $r[1,2] \subseteq s[1,2]$) and a participation with maximum cardinality 1 for the first component of $s$ (that

https://doi.org/10.1017/S1471068410000104 Published online by Cambridge University Press

generates the KD $key(S) = \{1\}$). The mentioned ID and KD are a fragment of a set of CDs that is sufficient to show that the chase may not exist even if the initial database satisfies the dependencies. Let the initial database be $D = \{r(a,b), s(a,c)\}$. Although $D$ satisfies the KD, the chase rule for the ID generates a tuple $s(a,b)$, which triggers a (failing) KD chase rule application on $s(a,b)$ and $s(a,c)$. Therefore the chase for this database and constraints does not exist.

Since the chase may be of infinite size, it would seem that checking whether a chase exists is semi-decidable. Indeed, in the general case of IDs and KDs it is not known whether it is decidable to check whether the chase exists.

However, the following lemma shows that termination of the chase under CDs is decidable; we will then use it to state some of our results.

*Lemma 1*
Let $D$ be a database for a relational schema $\mathscr{R}$ and $\Sigma$ a set of CDs over $\mathscr{R}$. Then, checking whether $chase_\Sigma(D)$ exists is decidable in time polynomial in the size of $D$.

*Proof*
We start by observing that the application of a *unary* ID (i.e., an ID that involves a single attribute) cannot cause a failure of the chase by violation of a KD: indeed, considering a generic unary ID $r_1[k_1] \subseteq r_2[k_2]$, the only possible violation of a KD due to the application of this ID is when we have the KD $key(r_2) = \{k_2\}$; however, such violation never causes a failure of the chase, since all values in the added tuple that are in positions different from $k_2$ are all fresh constants. Now, let us indicate with $\Sigma_R$ the IDs in $\Sigma$ that derive from is-a relations among relationships; they are IDs of the form $r_1[1,\ldots,n] \subseteq r_2[j_1,\ldots,j_n]$, where $j_1,\ldots,j_n$ is a permutation of $1,\ldots n$ and both $r_1$ and $r_2$ have arity $n$. It is immediately seen that:

(i) Facts in the chase of the form $r(c_1,\ldots,c_n)$, where $r$ is a relation belonging to the set $\mathscr{R}_R$ of $n$-ary relationships in the conceptual schema, contain
  - only nonfresh constants,
  - only fresh constants, or
  - exactly one nonfresh constant (possibly occurring more than once).

  No other case is possible. This can be shown by induction on the number of application of chase rules. Consider also that
  - Facts regarding (unary) predicates associated with entities may either contain a fresh or a nonfresh constant.
  - For facts regarding predicates associated with $n$-ary attributes, we have that the last position may be occupied by either a fresh or a nonfresh constant, and the first $n$ positions behave like a fact for a relation in $\mathscr{R}_R$ (i.e., they contain only nonfresh constants, only fresh constants, or exactly one nonfresh constant).

In the base case (no application), we only have facts in $D$, which only contain nonfresh constants. Suppose now, by inductive hypothesis, that, after $i$ applications of the chase rules, the facts are only of the forms mentioned above. The inductive step consists in showing that no new application of a chase rule produces facts that are not in one of the forms mentioned above. To see this, it suffices to verify this for all forms (1)–(11) of dependencies that may occur in CDs, as described in Section 4. This is immediate for (1)–(10). As for (11), consider that a KD rule can be applied on two tuples $t_1$ and $t_2$ for a relation $r \in \mathscr{R}_R$ in two cases:

- $t_1$ and $t_2$ both have in the position of the key the same nonfresh constant. In this case the inductive step immediately follows, by either a failure of the chase or the generation of a new tuple containing exactly one nonfresh constant (possibly occurring more than once).
- $t_1$ and $t_2$ both have in the position of the key the same fresh constant. The inductive step follows immediately, unless $t_1$ contains exactly one nonfresh constant, say $c$, and $t_2$ contains exactly one nonfresh constant, say $d$, with $d \neq c$, because then the KD rule could produce a tuple containing two different nonfresh constants. However, this case cannot occur. To see this, it suffices to show that if two tuples $t_1$ and $t_2$ for $r \in \mathscr{R}_R$ have a fresh constant in common, then they cannot have different nonfresh constants. This can, again, be shown by induction on the applications of chase rules for dependencies of the forms (1)–(11). Basically, the only way for tuples of relations in $\mathscr{R}_R$ to have fresh constants in common is to apply chase rules on dependencies of the forms (9)–(11).

  — With form (9), the application of the ID chase rule on a cycle of is-a relations between relationships may generate two tuples sharing a fresh constant. However, only permutations of the positions can take place, but the constants are unchanged.

  — Two applications of the ID chase rule on two different IDs of form (10) for the same entity and the same relationship but on two different components can generate two tuples sharing a fresh constant. However, all the other constants will also be fresh.

  — The application of a KD chase rule for a KD of form (11) is now trivially harmless by inductive hypothesis.

(ii) All facts of the form $r(c_1, \ldots, c_n)$, with $r \in \mathscr{R}_R$, which contain only nonfresh constants are obtained by applying (possibly several times) the ID chase rule for IDs in $\Sigma_R$ to facts in the initial database (constituted in turn by tuples containing only nonfresh constants).

(iii) By what stated in point (i) above, the only way of causing a failure in the chase construction (apart from violations of key constraints already in $D$) is to apply an ID in $\Sigma_R$ to a tuple having only nonfresh constants, thus introducing a (nonrepairable) violation of some KD due to the presence of another tuple having only nonfresh constants; in all other cases, every violation of a KD is repaired by applications of the KD chase rule.

This said, it follows that if there is no failure in $chase_{\Sigma_R}(D)$, there is no failure in $chase_{\Sigma}(D)$. It remains to check whether $chase_{\Sigma_R}(D)$ is finite: it is easily seen that it indeed cannot be infinite, since every tuple in $chase_{\Sigma_R}(D)$ is of the form $r(c_1, \ldots, c_n)$, with $r \in \mathscr{R}_R$, and where $c_1, \ldots, c_n$ are obtained by a permutation of $d_1, \ldots, d_n$, where the fact $r'(d_1, \ldots, d_n)$, with $r' \in \mathscr{R}_R$, is in the initial database $D$.

The maximum depth of $chase_{\Sigma_R}(D)$ is $W!$, where $W$ is the maximum arity of predicates in $\mathscr{R}$. It is also straightforward to see that the size of $chase_{\Sigma_R}(D)$ is polynomial in $|D|$ (size of $D$, i.e., number of tuples of $D$), and that $chase_{\Sigma_R}(D)$ can be constructed in time polynomial in $|D|$. By the above considerations, it is immediately seen that $chase_{\Sigma_R}(D)$ fails iff $chase_{\Sigma}(D)$ fails. The thesis follows. □

*Lemma 2*
Let $D$ be a database for a relational schema $\mathscr{R}$ and $\Sigma$ a set of CDs over $\mathscr{R}$ such that $chase_{\Sigma}(D)$ exists. Then $chase_{\Sigma}(D) \models \Sigma$.

*Proof*
Trivial, by the construction of Definition 5. □

The following lemma is a technical result that will be used in the proof of Theorem 1. Informally, it shows that the chase of a database $D$, when it exists, is a powerful tool for answering queries: for every solution $B$ (database that is a superset of the given incomplete database $D$ and that satisfies the constraints), there is a homomorphism that sends the chase of $D$ onto $B$. This result follows from the results in Fagin *et al.* (2005) and (Deutsch *et al.* 2008), but we provide a direct proof for the sake of completeness.

*Lemma 3*
Let $D$ be a database for a relational schema $\mathscr{R}$ and $\Sigma$ a set of CDs over $\mathscr{R}$ such that $chase_{\Sigma}(D)$ exists. Then, for every database $B$ for $\mathscr{R}$ such that $B \models \Sigma$ and $B \supseteq D$, we have that there exists a homomorphism from $chase_{\Sigma}(D)$ to $B$.

*Proof*
Similarly to what is done for the analogous result in Calì *et al.* (2004), we proceed by induction on the applications of the (ID or KD) chase rules. We define a homomorphism $\mu$ inductively, and we simultaneously show that for each relation $r$ of arity $n$ in $\mathscr{R}$, and each tuple $(c_1, \dots, c_n)$ constituted by elements in $\Gamma \cup \Gamma_f$, if $(c_1, \dots, c_n) \in r^{chase_{\Sigma}(D)}$, then $(\mu(c_1), \dots, \mu(c_n)) \in r^B$.

(1) **Base case**. After 0 applications of a chase rule, the constructed part of the chase coincides with $D$. Since $B \supseteq D$, the mapping $\mu$ that maps each constant in $D$ into itself is a homomorphism from the constructed part of the chase to $B$.

(2) **Inductive step**. *First case: the applied rule is the ID chase rule.* Suppose that in the application of the rule, we are inserting the tuple $t^* = (\alpha_1, \dots, \alpha_n)$ in $chase_{\Sigma}(D)$, where $r$ has arity $n$, $\alpha_i \in \Gamma_f$ for each $i \neq k$, $\alpha_k \in \Gamma \cup \Gamma_f$, and the tuple is inserted in $r^{chase_{\Sigma}(D)}$ because of the ID $w[j] \subseteq r[k]$ (other forms of IDs among those described in points (1)–(11) in Section 3 are dealt with similarly). Since we are applying the rule because of the dependency $w[j] \subseteq r[k]$, there is a tuple $t$ in $w^{chase_{\Sigma}(D)}$ such that $t[j] = \alpha_k$. By inductive hypothesis, there is a constant $c_k$ in $\Gamma$ such that $\mu(\alpha_k) = c_k$, and there is a tuple $t' \in w^B$ such that for each $i$, $t'[i] = \mu(t[i])$, with $t'[j] = \mu(\alpha_k) = c_k$. Because of the constraint $w[j] \subseteq r[k]$, and because $B$ satisfies the constraints, there is a tuple $t''$ in $r^B$ with $t''[k] = c_k$; let then $t'' = (c_1, \dots, c_n)$. Then, we set $\mu(\alpha_i) = c_i$ for each $i \neq k$, and we can conclude that $\mu(t^*) \in r^B$.

https://doi.org/10.1017/S1471068410000104 Published online by Cambridge University Press

*Second case: the applied rule is the KD chase rule.* By inductive hypothesis, there exists a homomorphism $\mu$ mapping the two tuples $t, t'$ on which the KD rule is applied into tuples $\mu(t)$ and $\mu(t')$ in $B$. Note that, since the KD rule is applicable to $t, t'$ and $B \models \Sigma$, we must have $\mu(t) = \mu(t')$. In the chase, $t$ and $t'$ are then replaced by a new tuple, say $t,''$ which contains (in the same positions) all the nonfresh constants of $t, t'$ and a subset of the fresh constants of $t, t'$ (some of which may disappear by the KD chase rule), but no new fresh constant. Therefore, $\mu$ trivially also maps $t,''$ as well as all other tuples in the chase, into facts of $B$. □

The following theorem is the main result of this section, and it characterizes the chase as a formal tool for query answering under KDs and IDs. In particular, the theorem states that the answers to a query $q$, posed on an incomplete database $D$ under a set $\Sigma$ of CDs, can be obtained by evaluating $q$ over the chase of $D$ w.r.t. $\Sigma$, $chase_\Sigma(D)$, and discarding the result tuples that contain at least one fresh value.

*Theorem 1*

Let $D$ be a database for a relational schema $\mathscr{R}$ and $\Sigma$ a set of CDs over $\mathscr{R}$ such that $chase_\Sigma(D)$ exists. Then, for every conjunctive query $q$ over $\mathscr{R}$, we have that $q^{[\Gamma]}(chase_\Sigma(D)) = ans(q, \Sigma, D)$.

*Proof*

The theorem is proved by considering a generic database $B$ such that $B \models \Sigma$ and $B \supseteq D$.

By Lemma 3 we derive the existence of a homomorphism $\mu$ that sends the facts of $chase_\Sigma(D)$ to facts of $B$; if $t \in q(chase_\Sigma(D))$, there is a homomorphism $\lambda$ from the atoms of $body(q)$ to $chase_\Sigma(D)$ that sends $head(q)$ to $t$; therefore, the composition $\lambda \circ \mu$ is a homomorphism from the atoms of $body(q)$ to $B$ that sends $head(q)$ to $t$, which proves $q(chase_\Sigma(D)) \subseteq ans(q, \Sigma, D)$, and, a fortiori, $q^{[\Gamma]}(chase_\Sigma(D)) \subseteq ans(q, \Sigma, D)$.

For the other inclusion, consider that $chase_\Sigma(D) \supseteq D$ and $chase_\Sigma(D) \models \Sigma$. Then, by Definition 1 we have that a tuple $t$ is a certain answer to $q$ in $D$ under $\Sigma$ only if it is an answer to $q$ in $chase_\Sigma(D)$ with no fresh constant; hence $q^{[\Gamma]}(chase_\Sigma(D)) \supseteq ans(q, \Sigma, D)$. □

Notice that Theorem 1 does not lead to an algorithm for query answering (apart from special cases), since the chase may have infinite size.

## 5 Answering queries by rewriting

In this section we present an efficient technique for query answering on incomplete data in the presence of CDs; such technique is based on *query rewriting*; in particular, the answers to a query are obtained by evaluating a new query, obtained by rewriting the original one according to the dependencies, over the initial incomplete data.

For the sake of simplicity, in the remainder of this section we shall disregard attributes from our treatment, since attributes are acyclic and therefore can be added without changing the results.

### 5.1 Query rewriting

Query answering under CDs can be decided by checking an initial segment of the chase of a database. We show that the certain answers to a CQ $q$ over a database $D$ can be computed by evaluating $q$ over the initial segment of the chase of $D$, whose size, defined by a maximum level $\delta_M$, depends on the query, on the dependencies, and on the size $\lambda_D$ of the largest connected part of the join graph of database $D$. The join graph of a database $D$ is an undirected graph that has as nodes the atoms of $D$ and has an arc $(\underline{A}, \underline{B})$ iff $\underline{A}$ and $\underline{B}$ share a constant.

*Theorem 2*

Let $\mathscr{R}$ be a relational schema, $\Sigma$ a set of CDs over $\mathscr{R}$, $q$ a conjunctive query over $\mathscr{R}$, and $D$ a database for which $chase_\Sigma(D)$ exists. Then, there is a number $\delta_M$ that depends on $q$, $\Sigma$, $\mathscr{R}$, and $\lambda_D$ such that for every tuple $t \in q^{[\Gamma]}(chase_\Sigma(D))$, there exists a homomorphism $\mu$ sending $body(q)$ to facts of $chase_\Sigma(D)$ and $head(q)$ to $t$ such that all the atoms in $\mu(body(q))$ are in the first $\delta_M$ levels of $chase_\Sigma(D)$.

*Proof*

First of all, we introduce the *chase forest* for $chase_\Sigma(D)$ given a database $D$ and a set of CDs $\Sigma$. The nodes of the forest are the atoms in $chase_\Sigma(D)$, and there is an arc $(\underline{A}_1, \underline{A}_2)$ iff $\underline{A}_2$ is generated from $\underline{A}_1$ by an application of the ID chase rule. The roots in the forest are the atoms in $D$, and they are at level 0. If there is an arc $(\underline{A}_1, \underline{A}_2)$ and $\underline{A}_1$ is at level $\ell$, then $\underline{A}_2$ is at level $\ell + 1$. In order to carry on the proof, we now prove that a constant can be propagated in the chase for at most a fixed number of levels that does not depend on $D$.

*Lemma 4*

Let $D$ be a database for a relational schema $\mathscr{R}$, $\Sigma$ a set of CDs over $\mathscr{R}$ such that $chase_\Sigma(D)$ exists, and $q$ a conjunctive query over $\mathscr{R}$. Let $a$ be a constant in $\Gamma$ occurring in an atom in $D$. Then $a$ never occurs in any fact with level greater than $\delta_D = \delta_C \cdot \lambda_D$ in $chase_\Sigma(D)$, where $\delta_C = |\mathscr{R}| \cdot (1 + |\mathscr{R}| \cdot W!)$.

*Proof*

We start by considering the IDs. First, observe that, in a set of CDs, the only nonunary IDs in $\Sigma$ are the IDs encoding is a arcs between relationships (which are full-width IDs) and the IDs regarding attributes of a relationship. Clearly, $a$ can be propagated to other atoms by applications of an ID chase rule, starting from the atom $\underline{\theta} \in D$ in which it occurs, then from the atom generated from $\underline{\theta}$ by the application, and so forth. The propagation can be done for up to $|\Sigma|$ more levels if there are no cycles in the IDs, but also for more, if there are cycles.

Whenever there is an application of an $n$-ary ID ($n \geqslant 2$) on an atom $\underline{A}$, the generated atom $\underline{A}'$ contains a permutation of the constants occurring in $\underline{A}$; both the involved predicates have the same arity $n$ (except in the case of an ID regarding attributes of a relationship, where one predicate has arity $n+1$, but the $(n+1)$-th argument is never used in the IDs). Then, a sequence of consecutive applications of $n$-ary IDs can go on for at most $n! \cdot |\mathscr{R}|$ levels, since there are $n!$ possible permutations of the constants in $\underline{A}$ and there are at most $|\mathscr{R}|$ relations involved in $n$-ary IDs. All constants occurring in $\underline{A}$ (except at most the last one, if $\underline{A}$ regards an attribute of a relationship) are propagated throughout the sequence.

All other applications regard unary IDs. At least one of the two predicates involved in a unary ID must be unary, and the only way to retain $a$ in a unary atom is that it be of the form $e(a)$, where $e$ is a unary predicate; clearly such fact can be generated only once in the chase, and there are at most $|\mathscr{R}|$ unary predicates in $\mathscr{R}$.

Any path in the chase starting from $\underline{\theta}$ consists of sequences of consecutive applications of $n$-ary IDs ($n \geqslant 2$) interleaved by applications of unary IDs. According to the previous considerations, there can be at most $|\mathscr{R}|+1$ sequences of consecutive applications of $n$-ary IDs (with $n \geqslant 2$ and $n \leqslant W$). Given the maximum lengths of such sequences, $a$ can be propagated for at most $\delta_C = |\mathscr{R}| \cdot (1 + |\mathscr{R}| \cdot W!)$.

We now consider the KDs. To prove the claim, we first state the following lemma.

*Lemma 5*

Let $\underline{A}$ be the first atom (of the form $r(\ldots, z_0, \ldots)$, where $r$ is $n$-ary, $n \geqslant 2$) in which a constant $z_0 \in \Gamma \cup \Gamma_f$ occurs, with $\ell = \mathit{level}(\underline{A}) > \delta_C$. Let $\underline{B}$ be the closest predecessor of atom $\underline{A}$ of the form $e(w_0)$ ($e$ unary). Let $\underline{B}'$ be an atom of the form $e(z_1)$, $z_1 \in \Gamma \cup \Gamma_f$, with $\mathit{level}(\underline{B}') > \ell + \delta_C$ such that there is an atom $\underline{C}$ of the form $e'(z_0)$ ($e'$ unary) in the path between $\underline{A}$ and $\underline{B}'$. Then no constant occurring in $\underline{A}$ other than $z_0$ occurs in any of the descendants of $\underline{B}'$.

*Proof*

Atom $\underline{C}$ may well have a child (or a descendant obtained by consecutive applications of the ID chase rule for nonunary IDs from the child) $\underline{D}$ of the form $r'(\ldots, z_1, \ldots)$ such that it agrees on the key of $r'$ (on value $z_1$) with some descendant $\underline{D}'$ of $\underline{B}'$ of the same form, so that the constants in $\underline{D}$ (possibly including $z_0$) will replace the corresponding constants of $\underline{D}'$ in all the descendants of $\underline{B}'$. Note that $\underline{B}'$ is necessarily a descendant of $\underline{C}$ with the same constants as $\underline{D}$. This shows that $z_0$ may well occur in some descendant of $\underline{B}'$. Let us indicate with $z_0'$ the constant that is replaced by $z_0$ after the application of the KD chase rule. Assume, by contradiction, that one of the constants in $\underline{A}$ other than $z_0$ occurs in some descendant of $\underline{B}'$. Then, there must be a descendant $\underline{A}'$ of $\underline{B}'$ of the form $r(\ldots, z_0', \ldots)$ that, once $z_0'$ is replaced by $z_0$, fires the application of a KD chase rule between $\underline{A}$ and $\underline{A}'$. There are two cases: (i) $\underline{A}'$ generates $\underline{D}'$ via a sequence of nonunary IDs. Then $z_1$ is replaced by $w_0$, then the subtree rooted in $\underline{B}'$ gets to have the same root as the subtree rooted in $\underline{B}$ and therefore it disappears as a consequence of the KD application. (ii) $\underline{A}'$ is a descendant of $\underline{C}'$ along a path that contains at least an application of the ID chase rule for a unary ID, where $\underline{C}'$ is obtained from $\underline{B}'$ by the same sequence of applications of ID chase rules as those generating $\underline{C}$ from $\underline{B}$. Again, the KD chase rule makes $\underline{C}'$ become equal to $\underline{C}$, therefore the whole subtree rooted in $\underline{C}'$ disappears, as easily seen, as above. □

Consider the proof of Lemma 5 and assume $z_0 \in \Gamma$. Then, after at most $\delta_C$ levels $z_0$ will not appear together with any of the other constants in $\underline{A}$. Also, $z_0$ cannot be propagated indefinitely in the chase by applications of ID chase rules, since this requires using $z_0$ with a unary predicate, which can be done only once per unary predicate. However, if $z_0$ appears in an atom in $D$ together with another constant

$c$, then $c$ could appear together with $z_0$ in a descendant of $\underline{B}'$, and propagate through further $\delta_C$ levels. By the same principle, this can go on for every sequence of constants $c_1, \ldots, c_n$ such that $c_i$ occurs in the same atom in $D$ together with $c_{i+1}$. Since the maximum sequence of this kind can have length $|\lambda_D|$, and the sequences in $D$ are not altered by the chase construction, the claim follows. □

Lemma 4 is the key property for stopping the construction of the chase at a given level $\delta_M$ without altering query answering. We first prove the claim for the simple but important subclass of conjunctive queries called nonboolean (i.e., with at least one distinguished variable) connected queries. A set of atoms $n$ is *connected* if the undirected graph $(n, \mathscr{A})$ is connected, where $n$ is the set of nodes, and $\mathscr{A}$ is the set containing exactly all arcs between any two atoms in $n$ that share a variable or a constant. A CQ $q$ is *connected* if $body(q)$ is. Every maximal subset of $body(q)$ that is connected is called a *connected part* of $q$. Assume $\mu$ is a homomorphism sending $head(q)$ to a nonempty tuple $t$ of constants in $\Gamma$ and $body(q)$ to atoms of $chase_\Sigma(D)$. Since the query has at least one distinguished variable, then there is at least one atom $\underline{A}$ in $body(q)$ such that $\mu(\underline{A})$ contains a constant $c_1$ of $t$, which then is in $\Gamma$. By Lemma 4, the constants in $\Gamma$ cannot occur at levels greater than $\delta_D$; then $level(\mu(\underline{A})) \leqslant \delta_D$. If a query is connected and nonboolean, then among the other body atoms there is at least another atom $\underline{A}'$ sharing a variable with $\underline{A}$, and thus such that $\mu(\underline{A}')$ shares a constant with $\mu(\underline{A})$. Note now that $\mu(\underline{A})$ contains $c_1$ plus possibly other constants. If such constants are in $\Gamma$, then also $\mu(\underline{A}')$ has a level at most $\delta_D$. Else, they are all fresh and have been created in the subtree rooted in the closest unary predecessor $\underline{B}$ of $\mu(\underline{A})$; $\underline{B}$ has the form $e_1(c_1)$. Now we show that all the constants different from $c_1$ (say, $z_1, \ldots, z_n$) in $\mu\underline{A}$ occur within the first $\delta_C$ levels of $\mu\underline{A}$, and therefore $\mu(\underline{A}')$ also occurs at a level at most $level(\mu\underline{A}) + \delta_C$. To see this, we simply reapply Lemma 5 by considering $\mu\underline{A}$ alone as the starting database for the subsequent propagation of constants. Indeed, for $1 \leqslant i, j \leqslant n$, the longest path from an atom containing $z_i$ (but not $z_j$) to an atom containing $z_j$ (but not $z_i$) in the join graph is 1. This process can be iterated for all the remaining atoms in the query. Since the size of the longest path in the graph of $q$ is $|q|$, it follows that all the images of the atoms of the query are in the first $\delta_M = \delta_D + \delta_C \cdot (|q| - 1)$ levels.

If the query is not connected, but each connected part is nonboolean, the same argument as before applies to each connected part, with the same final $\delta_M$.

If the query has at least a boolean connected part, we can reason as follows. Let $A$ be the atom in the connected part whose image $\mu(\underline{A})$ is at the lowest level among the query atoms. If $level(\mu(\underline{A})) > \delta_D$, then there is another homomorphism $\mu'$ sending $body(q)$ to atoms of $chase_\Sigma(D)$ such that $level(\mu'(\underline{A})) \leqslant \delta_D$, because all types occur within the first $\delta_D$ levels, where two atoms have the same type if they share the same predicate and agree on all the positions where a constant of $\Gamma$ appears. With the same argument as before, all the images via $\mu'$ are at a level at most $\delta_M$. □

The previous theorem suggests a naive strategy for query answering: first, compute the initial segment of $chase_\Sigma(D)$, i.e., its first $\delta_M$ levels, and then evaluate the query $q$ on such a segment. To do that, we also need the following Lemma.

https://doi.org/10.1017/S1471068410000104 Published online by Cambridge University Press

*Lemma 6*
Consider the application of a KD chase rule on two atoms $\underline{A_1}$ and $\underline{A_2}$ with $level(\underline{A_1}) = \ell_1 > \delta_D$ and $level(\underline{A_2}) = \ell_2 > \delta_D$. Consider also all subsequent applications of KD chase rules before the next application of an ID chase rule. Then, after all these applications, no atom in the chase is affected that has level lower than $\min\{\ell_1, \ell_2\} - \delta_C$.

*Proof*
By definition of the chase, when a KD chase rule is applied, the affected constants are the more recent ones in the chase construction. Then, it easily follows that they may only occur at most $\delta_C$ levels before $\min\{\ell_1, \ell_2\}$. Indeed, $\underline{A_1}$ and $\underline{A_2}$ have at least a constant in common. Two cases are possible: (i) they share a constant in $\Gamma$, therefore they may only occur within the first $\delta_D$ levels by Lemma 5, against the hypotheses; (i) they share a constant in $\Gamma_f$. In the latter case, they have a common unary predecessor $\underline{A_0}$ within $\delta_C$ levels before $\min\{\ell_1, \ell_2\}$. In this case, the replacement of constants has an impact only on the subtree $T$ rooted in $\underline{A_0}$ since all other constants in $T$ are by construction newer than the one occurring in $\underline{A_0}$. Ditto for the subsequent applications. □

By Lemma 6, it is immediate to see that the application of the KD chase rule does not affect any facts whose depth is smaller by at least $\delta_C$ levels than the level of the facts involved in the KD; therefore, to compute the first $\delta_M$ levels of $chase_\Sigma(D)$ means to apply the chase rules of Definition 5 until no chase rule is applicable on facts at a level smaller than $\delta_M + \delta_C$. However, it is easy to see that such a strategy would not be efficient in real-world cases, where $D$ has a large size. Our plan of attack is then to rewrite $q$ according to the CDs on the schema and on the size $\lambda_D$ of the largest connected part of the join graph, and then to evaluate the rewritten query over the initial data. This turns out to be more efficient in practice, if $\lambda_D$ is bounded or known to be reasonably small, since it does not involve the entire database $D$ in the query processing, except for the last evaluation step, so most of the computation is kept at the intensional level. In particular, the rewritten query is expressed in Datalog, and it is the union of two sets of rules, denoted $\Pi^{\Sigma_I}$ and $\Pi^{\Sigma_K}$, which take into account IDs and KDs respectively, plus a set of rules $\Pi^{eq}$ that simulates equality. Finally, function symbols present in the rules will be eliminated to obtain a Datalog rewriting.

Consider a relational schema $\mathscr{R}$ with a set $\Sigma$ of CDs, with $\Sigma = \Sigma_I \cup \Sigma_K$, where $\Sigma_I$ and $\Sigma_K$ are sets of IDs and KDs respectively. Let $q$ be a CQ over $\mathscr{R}$; we construct $\Pi^{eq}$, $\Pi^{\Sigma_I}$ and $\Pi^{\Sigma_K}$ in the following way.

**Encoding equalities.** We introduce a binary predicate $eq/2$ that simulates the equality predicate; to enforce reflexivity, symmetry and transitivity respectively, we introduce in $\Pi^{eq}$ the rules

(a) $eq(X_i, X_i) \leftarrow r(X_1, \ldots, X_n)$ for all $r/n$ in $\mathscr{R}$ and for all $i \in \{1, \ldots, n\}$
(b) $eq(Y, X) \leftarrow eq(X, Y)$
(c) $eq(X, Z) \leftarrow eq(X, Y), eq(Y, Z)$

https://doi.org/10.1017/S1471068410000104 Published online by Cambridge University Press

Similar rules for encoding equalities are found, for instance, in Duschka and Levy (1997) and (Gottlob and Nash 2008).

**Encoding key dependencies.** For every KD $key(r) = \{k\}$ (notice from Section 3 that in the case of CDs all keys are unary if the original EER schema contains no attributes), with $R$ of arity $n$, we introduce in $\Pi^{\Sigma_K}$ the rule

$$\begin{aligned} eq(X_i, Y_i) \quad \leftarrow \quad & r(X_1, \ldots, X_{k-1}, X_k, X_{k+1}, \ldots, X_n), \\ & r(Y_1, \ldots, Y_{k-1}, Y_k, Y_{k+1}, \ldots, Y_n), eq(X_k, Y_k) \end{aligned}$$

for all $i$ s.t. $1 \leqslant i \leqslant n$, $i \neq k$.

**Encoding inclusion dependencies.** The encoding of a set $\Sigma_I$ of IDs into a set $\Pi^{\Sigma_I}$ of rules is done in two steps. Similarly to (Calì *et al.* 2001; Calì 2003), every ID is encoded by a logic programming rule $\Pi^{\Sigma_I}$ with function symbols, appearing in Skolem terms that replace existentially quantified variables in the head of the rules; intuitively, they mimic the fresh constants that are added in the construction of the chase. We consider the four cases that are possible for an ID $\sigma$ in a set of CDs coming from an EER schema without attributes:

(1) $\sigma$ is of the form $r_1[1] \subseteq r_2[1]$, with $r_1/1, r_2/1$: we add to $\Pi^{\Sigma_I}$ the rule $r_2(X) \leftarrow r_1(X)$.
(2) $\sigma$ is of the form $r_1[k] \subseteq r_2[1]$, with $r_1/n$, $r_2/1, 1 \leqslant k \leqslant n$: we add to $\Pi^{\Sigma_I}$ the rule $r_2(X_k) \leftarrow r_1(X_1, \ldots, X_n)$.
(3) $\sigma$ is of the form $r_1[1, \ldots, n] \subseteq r_2[j_1, \ldots, j_n]$, with $r_1/n$, $r_2/n$, and where $(j_1, \ldots, j_n)$ is a permutation of $(1, \ldots, n)$: we add to $\Pi^{\Sigma_I}$ the rule $r_2(X_{j_1}, \ldots, X_{j_n}) \leftarrow r_1(X_1, \ldots, X_n)$.
(4) $\sigma$ is of the form $r_1[1] \subseteq r_2[k]$, with $r_1/1$, $r_2/n, 1 \leqslant k \leqslant n$: we add to $\Pi^{\Sigma_I}$ the rule $r_2(f_{\sigma,1}(X), \ldots, f_{\sigma,k-1}(X), X, f_{\sigma,k+1}(X), \ldots, f_{\sigma,n}(X)) \leftarrow r_1(X)$.

Note that in (4) we have used subscripts of the form $\sigma, j$ so as to indicate that for every dependency and for every attribute of $r_2$ there is a different function symbol.

*Example 6*
Consider the dependencies that do not involve attributes ($\sigma_4$–$\sigma_{13}$) from Example 2. They can be encoded as follows.

$$\begin{array}{rrcl}
\sigma_4: & \mathsf{employee}(X) & \leftarrow & \mathsf{works_in}(X, Y) \\
\sigma_5: & \mathsf{dept}(Y) & \leftarrow & \mathsf{works_in}(X, Y) \\
\sigma_6: & \mathsf{manager}(X) & \leftarrow & \mathsf{manages}(X, Y) \\
\sigma_7: & \mathsf{dept}(Y) & \leftarrow & \mathsf{manages}(X, Y) \\
\sigma_8: & \mathsf{employee}(X) & \leftarrow & \mathsf{manager}(X) \\
\sigma_9: & \mathsf{works_in}(X, Y) & \leftarrow & \mathsf{manages}(X, Y) \\
\sigma_{10}: & \mathsf{works_in}(X, f_{\sigma_{10},2}(X)) & \leftarrow & \mathsf{employee}(X) \\
\sigma_{11}: & \mathsf{manages}(X, f_{\sigma_{11},2}(X)) & \leftarrow & \mathsf{manager}(X) \\
\sigma_{12}: & eq(Y_1, Y_2) & \leftarrow & \mathsf{works_in}(X_1, Y_1), \mathsf{works_in}(X_2, Y_2), eq(X_1, X_2) \\
\sigma_{13}: & eq(Y_1, Y_2) & \leftarrow & \mathsf{manages}(X_1, Y_1), \mathsf{manages}(X_2, Y_2), eq(X_1, X_2)
\end{array}$$

https://doi.org/10.1017/S1471068410000104 Published online by Cambridge University Press

**Query maquillage.** Since we need to deal with equalities among values in a uniform way, we need some maquillage (that we call *equality maquillage*) on $q$: replace every term $t$ in $body(q)$, with a new variable $X$ not occurring elsewhere in $q$, and add (as a conjunct) to $body(q)$ the atom $eq(X,t)$. Henceforth, we shall denote with $q_{eq}$ the query after the equality maquillage. For example, the query $q(X) \leftarrow r(X,c,Y), s(Y)$ becomes $q(X) \leftarrow r(A,B,C), s(D), eq(A,X), eq(B,c), eq(C,Y), eq(D,Y)$.

We shall now state that the encoding of CDs by means of the above rules captures the correct manipulation of facts that is done in the chase (that, we remind the reader, represents the inference of information done starting from the initial data and the CDs, under the sound semantics). In order to do that, in Theorem 3 below, we first need to introduce a few auxiliary constructions and lemmata.

We introduce a variant of the chase with equality predicates, denoted $chase^{eq}{}_{\Sigma}(D)$, which is built as follows from a database $D$ and a set of CDs $\Sigma$.

(1) Add all atoms of the form $eq(c,c)$, at level 0, where $c$ is a constant occurring in $D$.
(2) Include all the facts in $D$ and proceed as for $chase_{\Sigma}(D)$, but
   (a) A KD is applicable if there is a key constraint $key(r) = \{k_1, \ldots, k_n\}$ and the chase result constructed so far contains the facts $r(t), r(t')$, and $eq(\alpha_1, \beta_1), \ldots, eq(\alpha_n, \beta_n)$, with $\alpha_i = t[k_i]$ and $\beta_i = t'[k_i]$. When applying the KD rule, instead of merging tuples by replacing the two constants $\alpha_i$ and $\beta_i$ by a combined symbol, add the atoms $eq(\alpha_i, \beta_i)$, $eq(\beta_i, \alpha_i)$ and all the $eq$ atoms that can be derived from the existing ones by transitivity; the level of these $eq$ atoms is the same as the lower of the two facts that fired the rule.
   (b) An ID rule is applicable if there is an ID $r[k_1, \ldots, k_n] \subseteq s[j_1, \ldots, j_n]$ such that the chase result constructed so far contains the fact $r(t)$ but there is no fact $s(t')$ such that, for every $i$ such that $1 \leqslant i \leqslant n$, $eq(t[k_i], t'[j_i])$ is in the chase result constructed so far. When applying the ID rule, add the atom $eq(\alpha, \alpha)$ for each new fresh constant $\alpha$ in the newly introduced fact; the level of $eq(\alpha, \alpha)$ is the same as the level of the new fact.
   (c) Whenever an atom of the form $eq(c_1, c_2)$ is added, where $c_1, c_2 \in \Gamma$, and $c_1 \neq c_2$, stop the chase procedure (the chase fails).

*Example 7*
Consider again the EER schema of Example 2 and the initial (incomplete) database $D = \{\mathsf{manager}(m), \mathsf{works\_in}(m,d)\}$ given in Example 4. Then $chase^{eq}{}_{\Sigma}(D)$ consists of $D$ plus the following facts:

- $eq(m,m)$, $eq(d,d)$ (constants at level 0)
- $\mathsf{employee}(m)$, $\mathsf{manages}(m,\alpha_1)$, $\mathsf{works\_in}(m,\alpha_1)$, $\mathsf{dept}(\alpha_1)$, where $\alpha_1$ is a fresh constant (applications of ID chase rules)
- $eq(\alpha_1, \alpha_1)$ (new fresh constants)
- $eq(\alpha_1, m)$, $eq(m, \alpha_1)$ (application of a KD chase rules)

It is straightforwardly seen that $chase_{\Sigma}(D)$ exists if and only if $chase^{eq}{}_{\Sigma}(D)$ exists. Clearly, as stated in the following lemma, an isomorphism can be established between the atoms in $chase^{eq}{}_{\Sigma}(D)$ and those in the least Herbrand model of the program consisting of $D$ plus the rules encoding IDs, KDs, and equality.

*Lemma 7*

Consider a database $D$ over a relational schema $\mathscr{R}$ with a set of CDs $\Sigma = \Sigma_I \cup \Sigma_K$, where $\Sigma_K$ and $\Sigma_I$ are sets of KDs and IDs respectively, such that $chase_{\Sigma}(D)$ exists. Let $\Pi$ be the program $\Pi^{\Sigma_I} \cup \Pi^{\Sigma_K} \cup \Pi^{eq} \cup D$ and $M$ its least Herbrand model. Then, there is an isomorphism $\mu : \Gamma \cup \Gamma_f \rightarrow U_{\Pi}$, where $U_{\Pi}$ is the Herbrand universe[2] of $\Pi$, such that: (i) $\mu(chase^{eq}{}_{\Sigma}(D)) = M$; (ii) if $\alpha \in \Gamma_f$ then $\mu(\alpha)$ is a Skolem ground term in $U_{\Pi}$.

*Proof*

We exhibit the construction of a homomorphism with the desired properties. The construction will be inductive on the applications of the immediate consequence operator in the construction of $M$. We start from $D$, and we take the identity isomorphism mapping $D$ (as a subset of $chase^{eq}{}_{\Sigma}(D)$) into $D$ (as a subset of $M$). Now we consider the following cases of application of the immediate consequence operator, on different kind of rules.

(1) *Rule in* $\Pi^{\Sigma_I}$. Assume we are adding a fact $s(\bar{t}_s)$ because of a rule $\rho$ of the form $s(\cdot) \leftarrow r(\cdot)$ encoding a dependency $\sigma$ of the form $r[\cdot] \subseteq s[\cdot]$, where $r(\bar{t}_r)$ is a fact in the part $M^*$ of $M$ constructed at a certain point. Since, by induction hypothesis, $\mu$ (so far) maps a subset of $chase^{eq}{}_{\Sigma}(D)$ to $M^*$, we take $\mu^{-1}(r(\bar{t}_r))$, which is of the form $r(\bar{u}_r)$: by application of the ID chase rule on $\sigma$ (encoded by $\rho$), we get the addition of a fact $s(\bar{u}_s)$. Now extend $\mu$ by adding to it $\{\bar{u}_s[i] \rightarrow \bar{t}_s[i]\}$ for every $i$ such that $\bar{u}_s[i]$ is a newly introduced fresh constant (or, equivalently, the corresponding argument in $\rho$'s head contains a Skolem term).

(2) *Rule in* $\Pi^{\Sigma_K}$. The construction is the same as above, where the added fact in $M^*$ is of the form $eq(t_1, t_2)$, with $\{t_1, t_2\} \subseteq U_{\Pi}$, and the one in $chase^{eq}{}_{\Sigma}(D)$ is of the form $eq(u_1, u_2)$, with $\{u_1, u_2\} \subseteq \Gamma \cup \Gamma_f$.

(3) *Rule in* $\Pi^{eq}$. It is straightforwardly seen that rules in $\Pi^{eq}$ introduce equality atoms, whose corresponding atoms in $chase^{eq}{}_{\Sigma}(D)$ are introduced by enforcing reflexivity, symmetry and transitivity of the predicate $eq$, as described in the construction of $chase^{eq}{}_{\Sigma}(D)$. The homomorphism $\mu$ is extended accordingly in an obvious way.

It is immediate to see that the isomorphism $\mu$ constructed as above is such that values in $\Gamma_f$ are mapped to Skolem terms (containing function symbols) and vice-versa, and that $\mu(chase^{eq}{}_{\Sigma}(D)) = M$. □

The previous lemma shows an isomorphism between the chase with equalities and the least Herbrand model of the program comprising the rules for IDs, KDs, equalities, and the database. Notice that this result holds for general IDs and KDs,

[2] Usually, the Herbrand universe is constructed with respect to a language, but often we can talk about the Herbrand universe of a logic program, intending the Herbrand universe constructed with the constants and function symbols present in that program. The same holds for the notion of Herbrand base.

and not only for CDs: in fact, arbitrary IDs and KDs can be encoded in the same way we did for CDs.

We then use Lemma 7 to extend the notion of level to the atoms of the least Herbrand model: the level of such an atom is defined as the level of the corresponding (via the isomorphism) atom in the chase with equalities.

Next, we show that, if we exclude the tuples containing fresh constants, the answers to a query over the chase coincide with the answers to the query after maquillage over the chase with equalities.

*Lemma 8*

Consider a conjunctive query $q$ over a relational schema $\mathscr{R}$ with a set of CDs $\Sigma = \Sigma_I \cup \Sigma_K$, where $\Sigma_K$ and $\Sigma_I$ are sets of KDs and IDs respectively, and a database $D$ for $\mathscr{R}$, such that $chase_\Sigma(D)$ exists. Then the tuples in $q_{eq}^{[\Gamma]}(chase^{eq}{}_\Sigma(D))$ coincide with those in $q^{[\Gamma]}(chase_\Sigma(D))$.

*Proof*

By construction of $chase^{eq}{}_\Sigma(D)$, if we eliminate all atoms of the form $eq(\alpha, \beta)$ from $chase^{eq}{}_\Sigma(D)$ and replace $\alpha$ with $\beta$ (or $\beta$ with $\alpha$, provided that the replacing one is the fresh constant that lexicographically comes first), we obtain $chase_\Sigma(D)$. We call this process *equality elimination*. Suppose that tuple $t$ consisting of nonfresh constants is in $q_{eq}(chase^{eq}{}_\Sigma(D))$. Then there exists a homomorphism $\mu$ sending $body(q_{eq})$ to atoms of $chase^{eq}{}_\Sigma(D)$ and $head(q_{eq})$ to $t$. By applying equality elimination to $\mu(body(q_{eq}))$ we then obtain atoms in $chase_\Sigma(D)$. These are, in turn, an image for a homomorphism $\mu'$ from $body(q)$ to atoms of $chase_\Sigma(D)$. This can be seen as follows. Consider an atom of the form $eq(X, u)$ in $body(q_{eq})$ such that $\mu(body(q_{eq})) = eq(c_1, c_2)$, where $X$ is a variable, $u$ a term, and $c_1, c_2 \in \Gamma \cup \Gamma_f$. Each time an atom of the form $eq(c_1, c_2)$ is eliminated by equality elimination from $\mu(body(q_{eq}))$, remove $eq(X, u)$ from $q_{eq}$ and replace in it all occurrences of the variable $X$ with the term $u$. At each step of the $eq$ elimination process, the two structures are isomorphic; at the end, $q_{eq}$ is transformed into a variant of $q$ (i.e., the same as $q$ modulo variable renaming), which proves that $q$ is isomorphic to the result of the equality elimination applied to $\mu(body(q_{eq}))$, i.e., there is the homomorphism $\mu'$ we were looking for. By construction of $q_{eq}$, if $t$ contains no fresh constant, then $\mu'$ necessarily maps $head(q)$ to $t$.

For the other inclusion, consider a homomorphism $\mu'$ sending $body(q)$ into atoms of $chase_\Sigma(D)$ and $head(q)$ into $t$. If the atoms in $\mu'(body(q))$ are in $D$, these are necessarily also in $chase^{eq}{}_\Sigma(D)$, so all non-$eq$ atoms in $body(q_{eq})$ can also be mapped to them by some homomorphism $\mu$; then, the $eq$ atoms require the equality of constants in $D$, which are necessarily present in $chase^{eq}{}_\Sigma(D)$. Then $t$ is also an answer in $q_{eq}(chase^{eq}{}_\Sigma(D))$. By construction of the $chase^{eq}{}_\Sigma(D)$, for every fact $f$ in $chase_\Sigma(D)$ there is a subset $S$ of $chase^{eq}{}_\Sigma(D)$, containing only one non-$eq$ fact $f'$, such that equality elimination on $S$ yields $f$; we say that $f'$ corresponds to $f$. If some atom in $\mu'(body(q))$ is not in $D$, it may have been generated by an ID rule or by a KD rule. In the case of an application of an ID rule on a fact $f$ in the chase, then there is a corresponding fact $f' \in chase^{eq}{}_\Sigma(D)$ on which the same application is made; note that no tuple merging caused by KD rules in the chase causes new applications of an ID rule. For a KD rule, in the chase an application instantiates fresh constants to other constants from two starting tuples; in the chase with equalities, the new tuple is not generated, but the two starting tuples remain, and *eq* atoms are generated for all merged constants. This means that if an atom in $q$ is mapped into such a merged fact, the corresponding (non-*eq*) atom in $q_{eq}$ can still be mapped into any of the two starting tuples. By construction of $q_{eq}$, the body of $q_{eq}$ contains one *eq* atom per term in $q$, so that each such term can be equalled to the replacing constant in the KD rule application (or be left unchanged by mapping the *eq* atom to one that equals the term to itself). □

With an argument similar to the one used in the proof of Theorem 2, it can be shown that, also for the chase with equality, $\delta_M$ levels are sufficient for query answering. This result is stated below as a corollary of Theorem 2.

*Corollary 1*
Let $D$ be a database for a relational schema $\mathscr{R}$, $\Sigma$ a set of CDs over $\mathscr{R}$ such that $chase_\Sigma(D)$ exists, and $q$ a conjunctive query over $\mathscr{R}$. Then, for every tuple $t \in q^{[\Gamma]}(chase^{eq}{}_\Sigma(D))$, there exists a homomorphism $\mu$ sending $body(q)$ to facts of $chase^{eq}{}_\Sigma(D)$ and $head(q)$ to $t$ such that all the atoms in $\mu(body(q))$ are in the first $\delta_M$ levels of $chase^{eq}{}_\Sigma(D)$, where $\delta_M$ is as in Theorem 2.

Now we can show the main result of this subsection as a consequence of the previous results. This result validates our encoding of IDs, KDs and equalities into $\Pi^{\Sigma_I}$, $\Pi^{\Sigma_K}$, $\Pi^{eq}$ and the query maquillage that returns $q_{eq}$ from $q$. Indeed, if we put together $\Pi^{\Sigma_I}$, $\Pi^{\Sigma_K}$, $\Pi^{eq}$ and $q_{eq}$ into a program $\Pi_{q_{eq}}$, and we evaluate it over a set $D$ of ground atoms, discarding the answer tuples that contain function symbols, we get exactly the certain answers to $q$, evaluated over $D$ under $\Sigma_I \cup \Sigma_K$.

*Theorem 3*
Consider a conjunctive query $q$ over a relational schema $\mathscr{R}$ with a set of CDs $\Sigma = \Sigma_I \cup \Sigma_K$, where $\Sigma_K$ and $\Sigma_I$ are sets of KDs and IDs respectively, and a database $D$ for $\mathscr{R}$, such that $chase_\Sigma(D)$ exists. Let $\Pi$ be the set of Horn clauses $q_{eq} \cup \Pi^{\Sigma_I} \cup \Pi^{\Sigma_K} \cup \Pi^{eq}$ and let $\Pi^{ff}_{q_{eq}}(D)$ be the largest function-free subset of $\Pi_{q_{eq}}(D)$. Then $\Pi^{ff}_{q_{eq}}(D) = ans(q, \Sigma, D)$.

*Proof*
By Lemma 7, we know that, if we exclude the atoms with predicate $q_{eq}$, the least Herbrand model $M$ of $\Pi \cup D$ coincides with $chase^{eq}{}_\Sigma(D)$ modulo an isomorphism that sends the fresh constants into Skolem terms, and the nonfresh constants into themselves. Therefore, $\Pi_{q_{eq}}(D)$ coincides with the answers in $q_{eq}(chase^{eq}{}_\Sigma(D))$, modulo this isomorphism; moreover, $\Pi^{ff}_{q_{eq}}(D)$ coincides with $q^{[\Gamma]}_{eq}(chase^{eq}{}_\Sigma(D))$, since, because of the bijection, atoms with fresh constants correspond to atoms with Skolem terms, and vice versa.

By Lemma 8, we know that $q^{[\Gamma]}_{eq}(chase^{eq}{}_\Sigma(D)) = q^{[\Gamma]}(chase_\Sigma(D))$.

Finally, Theorem 1 guarantees that $q^{[\Gamma]}(chase_\Sigma(D)) = ans(q, \Sigma, D)$, which concludes the proof. □

The above result is crucial because it shows the correctness and completeness of the encoding of the constraints into logic programming rules.

https://doi.org/10.1017/S1471068410000104 Published online by Cambridge University Press

In the next subsection we show how to eliminate the function symbols from $\Pi$, thus obtaining a program expressed in pure Datalog.

### *5.2 Elimination of function symbols*

Now, we want to transform the set of rules $\Pi$ of Theorem 3 into another set which has pure Datalog rules without function symbols. The reason to do so is that in this way we can take advantage of efficient Datalog engines, while evaluating logic programs with function symbols would certainly be an overkill.

To do that, we adopt a strategy somehow inspired by the elimination of function symbols in the *inverse rules* algorithm (Duschka and Genesereth 1997) for answering queries using views. The problem here is more complicated, due to the fact that function symbols may be arbitrarily nested in the least Herbrand model of the program. The idea here is to rely on the fact that there is a finite number $\delta_M$ of levels in the chase that is sufficient to answer a query, as stated in Theorem 2. We shall construct a Datalog program that mimics *only* the first $\delta_M$ levels of the chase, so that the function symbols that it needs to take into account are nested up to $\delta_M$ times. The strategy is based on the "simulation" of facts with function symbols in the least Herbrand model of $\Pi \cup D$ (where $D$ is an initial incomplete database) by means of ad-hoc predicates that are annotated so as to represent facts with function symbols.

*Definition 6* (*Annotation, annotated predicate, annotated version of an atom*)
Let $\underline{A}$ be an atom of the form $r(t_1, \ldots, t_n)$, where every term $t_i$ is of the form $f_{i,1}(f_{i,2}(\ldots f_{i,m_i}(\theta_i)\ldots))$, every $f_{i,j}$ is a unary function symbol, and every $\theta_i$ is either a constant in $\Gamma \cup \Gamma_f$ or a variable. The sequence $\bar{\eta} = \eta_1, \ldots, \eta_n$, with $\eta_i = f_{i,1}(f_{i,2}(\ldots f_{i,m_i}(\bullet)\ldots))$, is called the *annotation* of $\underline{A}$. The new $n$-ary predicate $r^{\bar{\eta}}$ is called the *annotated predicate* for $\underline{A}$, and the function-free atom $r^{\bar{\eta}}(\theta_1, \ldots, \theta_n)$ is called the *annotated version* of $\underline{A}$.

*Example 8*
The annotated version of the atom $\mathsf{works\_in}(X, f_{\sigma_{10},2}(X))$ occurring in the head of rule $\sigma_{10}$ in Example 6 is $\mathsf{works\_in}^{\bullet, f_{\sigma_{10},2}(\bullet)}(X, X)$.

Now, to have a program that yields function-free facts as described above, we construct suitable rules that make use of annotated predicates. The idea here is that we want to take control of the nesting of function symbols in the least Herbrand model of the program, by explicitly using annotated predicates that represent facts with function symbols; this is possible since we do that only for the (ground) atoms that mimic facts that are in the first $\delta_M$ levels of the chase of the incomplete database. Here we make use of the fact, proved in Lemma 7, that the least Herbrand model of $\Pi^{\Sigma_I} \cup \Pi^{\Sigma_K} \cup \Pi^{eq} \cup D$ coincides with $chase^{eq}{}_{\Sigma}(D)$, modulo renaming of the Skolem terms into fresh constants. Therefore, we are able to transform a (part of a) chase into the corresponding (part of the) least Herbrand model.

To do so, we construct a "dummy chase," and transform it, in the following way.

*Definition 7* (*Dummy database, dummy chase, dummy chase rules*)
Consider a relational schema $\mathscr{R}$ with a set $\Sigma_I$ of IDs.

(1) Let $B$ be a database for $\mathscr{R}$ consisting of exactly one fact of the form $r(c_1,\ldots,c_n)$ for every relation $r/n \in \mathscr{R}$, where $c_1,\ldots,c_n$ are distinct constants such that no constant occurs in more than one fact[3]; $B$ is called the *dummy database* for $\mathscr{R}$.

(2) Let $chase^{\delta_M}_{\Sigma_I}(B)$ denote the initial segment of $chase_{\Sigma_I}(B)$ consisting of the first $\delta_M$ levels; $chase^{\delta_M}_{\Sigma_I}(B)$ is called the *dummy chase* for $\mathscr{R}$ and $\Sigma_I$.

(3) Let $\mathscr{H}$ be as $chase^{\delta_M}_{\Sigma_I}(B)$, but where each fact (possibly containing fresh constants) is replaced with the corresponding atom (possibly containing function symbols) in the least Herbrand model of $\Pi^{\Sigma_I} \cup B$; note that such a correspondence exists by Lemma 7, because without KDs, if we exclude the *eq* atoms, $chase_{\Sigma_I}(B)$ and $chase^{eq}{}_{\Sigma_I}(B)$ coincide.

(4) Let $\mathscr{H}'$ be as $\mathscr{H}$, but where every atom is replaced with its annotated version.

(5) We denote with $\Pi^{DC}$ the set of all rules of the form $\underline{A}'_2 \leftarrow \underline{A}'_1$ such that (a) there is an arc $(\underline{A}_1,\underline{A}_2)$ in $\mathscr{H}'$, and (b) by replacing every distinct constant with a distinct variable in $(\underline{A}_1,\underline{A}_2)$, we obtain $(\underline{A}'_1,\underline{A}'_2)$. The rules in $\Pi^{DC}$ are called *dummy chase rules*.

*Example 9*
Consider Example 2; in the dummy chase, we introduce, among the others, the fact $\mathsf{employee}(\mathsf{c})$. This fact generates, according to the ID $\sigma_{10}$ : $\mathsf{employee}[1] \subseteq \mathsf{works\_in}[1]$, the fact $\mathsf{works\_in}(c, f_{\sigma_{10},2}(c))$ (after the transformation of the fresh constants into Skolem terms). Its annotated version is $\mathsf{works\_in}^{\bullet, f_{\sigma_{10},2}(\bullet)}(c,c)$. Therefore, $\Pi^{DC}$ contains, among the others, the rule $\mathsf{works\_in}^{\bullet, f_{\sigma_{10},2}(\bullet)}(X,X) \leftarrow \mathsf{employee}^{\bullet}(X)$.

The dummy chase determines all possible nesting sequences of function symbols that may occur in the first $\delta_M$ levels of the least Herbrand model of the program $\Pi^{\Sigma_I} \cup \Pi^{\Sigma_K} \cup \Pi^{eq} \cup D$: only IDs generate function symbols, and the dummy chase produces all possible function symbol sequences that may occur for every relation.

We next show how to generate a new annotated, function-free program from $\Pi^{\Sigma_I} \cup \Pi^{\Sigma_K} \cup \Pi^{eq}$. Preliminarily, we need some notation: we denote with $\bar{X}[h]$ the $h$-th term of a sequence $\bar{X}$, and with $\bar{\eta}[h]$ the $h$-th element of an annotation $\bar{\eta}$ (which is in turn a sequence).

*Definition 8* (*Function-free rewriting for CDs*)
Consider a conjunctive query $q$ over a relational schema $\mathscr{R}$ with a set of CDs $\Sigma = \Sigma_I \cup \Sigma_K$, where $\Sigma_K$ and $\Sigma_I$ are sets of KDs and IDs respectively. Let $\Pi^{ba}$ be the set of all rules, called *base annotation rules*, of the form $r^{\bullet,\ldots,\bullet}(X_1,\ldots,X_n) \leftarrow r(X_1,\ldots,X_n)$ for every predicate $r \in \mathscr{R} \cup \{eq\}$.

We define $\Pi^{q,\Sigma}$ as the set of rules $\Pi^{DC} \cup \Pi^{ba}$ plus all possible rules of the form $p_0^{\bar{\eta}_0}(\bar{t}_0) \leftarrow p_1^{\bar{\eta}_1}(\bar{t}_1),\ldots,p_k^{\bar{\eta}_k}(\bar{t}_k)$ such that:

(1) There is a rule $p_0(\bar{t}_0) \leftarrow p_1(\bar{t}_1),\ldots,p_k(\bar{t}_k)$ in $\Pi^{\Sigma_K} \cup \Pi^{eq} \cup q_{eq}$.
(2) Each annotation element $\bar{\eta}_i[j]$ occurs in some rule in $\Pi^{DC}$.
(3) If $\bar{t}_i[j] = \bar{t}_{i'}[j']$ then $\bar{\eta}_i[j] = \bar{\eta}_{i'}[j']$.

[3] It does not matter whether they are fresh or nonfresh, since they will disappear at the end of the process.

Base annotation rules are just a convenient renaming that allows us to refer to the annotation $\bullet,\ldots,\bullet$ to capture also the facts in the database. Note that $\Pi^{\Sigma_I}$ is not included in the program since it is already encoded in $\Pi^{DC}$ in a function-free fashion.

*Example 10*
Consider the dependency

$$\sigma_{13} : eq(Y_1, Y_2) \leftarrow \mathsf{manages}(X_1, Y_1), \mathsf{manages}(X_2, Y_2), eq(X_1, X_2)$$

encoding the KD $key(\mathsf{manages}) = \{1\}$ from Example 2. Among the annotations occurring in $\Pi^{DC}$, we have $f_{\sigma_{10},2}(\bullet)$ and $\bullet$ (note that $\bullet$ necessarily does), as shown in Example 9. Then $\Pi^{q,\Sigma}$ will include, among others, the rules

$$
\begin{array}{l}
eq^{\bullet,\bullet}(Y_1, Y_2) \leftarrow \mathsf{manages}^{\bullet,\bullet}(X_1, Y_1), \mathsf{manages}^{\bullet,\bullet}(X_2, Y_2), eq^{\bullet,\bullet}(X_1, X_2)\\
eq^{\bullet,\bullet}(Y_1, Y_2) \leftarrow \mathsf{manages}^{f_{\sigma_{10},2}(\bullet),\bullet}(X_1, Y_1), \mathsf{manages}^{\bullet,\bullet}(X_2, Y_2), eq^{f_{\sigma_{10},2}(\bullet),\bullet}(X_1, X_2)\\
eq^{\bullet,\bullet}(Y_1, Y_2) \leftarrow \mathsf{manages}^{\bullet,\bullet}(X_1, Y_1), \mathsf{manages}^{f_{\sigma_{10},2}(\bullet),\bullet}(X_2, Y_2), eq^{\bullet,f_{\sigma_{10},2}(\bullet)}(X_1, X_2)\\
eq^{f_{\sigma_{10},2}(\bullet),\bullet}(Y_1, Y_2) \leftarrow \mathsf{manages}^{\bullet,f_{\sigma_{10},2}(\bullet)}(X_1, Y_1), \mathsf{manages}^{\bullet,\bullet}(X_2, Y_2), eq^{\bullet,\bullet}(X_1, X_2)\\
eq^{\bullet,f_{\sigma_{10},2}(\bullet)}(Y_1, Y_2) \leftarrow \mathsf{manages}^{\bullet,\bullet}(X_1, Y_1), \mathsf{manages}^{\bullet,f_{\sigma_{10},2}(\bullet)}(X_2, Y_2), eq^{\bullet,\bullet}(X_1, X_2)\\
eq^{\bullet,\bullet}(Y_1, Y_2) \leftarrow \mathsf{manages}^{f_{\sigma_{10},2}(\bullet),\bullet}(X_1, Y_1), \mathsf{manages}^{f_{\sigma_{10},2}(\bullet),\bullet}(X_2, Y_2), eq^{f_{\sigma_{10},2}(\bullet),f_{\sigma_{10},2}(\bullet)}\\
(X_1, X_2)\\
\vdots
\end{array}
$$

Now we can state our central theorem.

*Theorem 4*
Let $D$ be a database for a relational schema $\mathscr{R}$, $\Sigma$ a set of CDs over $\mathscr{R}$ such that $chase_\Sigma(D)$ exists, and $q$ a conjunctive query over $\mathscr{R}$. Then, $\Pi^{q,\Sigma}_{q_{eq}^{\bullet,\ldots,\bullet}}(D) = ans(q, \Sigma, D)$.

*Proof*
The proof is based on the fact that the least Herbrand model $M$ of $\Pi^{q,\Sigma} \cup D$ is a representation of the first $\delta_M$ levels of the least Herbrand model $M_f$ of $q_{eq} \cup \Pi^{eq} \cup \Pi^{\Sigma_I} \cup \Pi^{\Sigma_K} \cup D$. By Lemma 7, the first $\delta_M$ levels of $M_f$ are isomorphic with the first $\delta_M$ levels of $chase^{eq}{}_\Sigma(D)$. By Corollary 1, the (nonfresh) answers to $q_{eq}$ over the first $\delta_M$ levels of $chase^{eq}{}_\Sigma(D)$ coincide with those found over the whole $chase^{eq}{}_\Sigma(D)$. By Lemma 8, the (nonfresh) answers to $q_{eq}$ over $chase^{eq}{}_\Sigma(D)$ coincide with the (nonfresh) answers to $q$ over $chase_\Sigma(D)$, which, by Theorem 1, coincide with $ans(q, \Sigma, D)$. Hence, to prove the thesis, we need to show that there is a correspondence between the facts in $M$ and those in the first $\delta_M$ levels of $M_f$.

We then represent the atoms in $M$ and those in the first $\delta_M$ levels of $M_f$ as two isomorphic structures. Consider therefore the atoms in $M_f$ as being disposed in levels (as in the corresponding chase with equalities). Every two atoms corresponding to an ID rule application are connected by an arc. An $eq$ atom has an incoming arc for each corresponding atom in the first rule (in $\Pi^{\Sigma_K}$ or $\Pi^{eq}$) that produced it via the immediate consequence operator. If we exclude $eq$ atoms, $M_f$ is a forest whose roots are the atoms in $D$; if we include the $eq$ atoms, we have a directed acyclic graph, since $eq$ atoms may have several parents. We now show that, for each atom $\underline{A}$ of the form $p(\theta_1, \ldots, \theta_n)$ in the first $\delta_M$ levels of $M_f$ there is an atom $\underline{B}$ of the

https://doi.org/10.1017/S1471068410000104 Published online by Cambridge University Press

form $p^{\eta_1,\dots,\eta_n}(c_1,\dots,c_n)$ in $M$, where each $\eta_i$ is the annotation element corresponding to $\theta_i$ and $c_i$ its innermost constant. Consider all the ancestors of $\underline{A}$ in $M_f$.

If $p$ is not the $eq$ predicate, there is a path $\underline{A}_0,\dots,\underline{A}_m = \underline{A}$ in $M_f$, such that $\underline{A}_i$ is at level $i$ and $\underline{A}_i$ is $\underline{A}_{i+1}$'s parent. We prove the claim by induction. As base case, we show that there is an atom $\underline{B}_0$ in $M$ corresponding to $\underline{A}_0$ and an annotation corresponding to $\underline{A}_0$'s predicate and terms in $\Pi^{q,\Sigma}$; but this is obvious, since $\underline{A}_0 \in D$ and all atoms in $D$ are also in $M$; besides, they also exist in $M$ with a $^{\bullet,\dots,\bullet}$ annotation, because of the base annotation rules in $\Pi^{ba}$. As inductive step, assume the claim holds for all $\underline{A}_j$ with $j \leqslant i$ and an annotation corresponding to $\underline{A}_i$'s predicate and terms is in $\Pi^{q,\Sigma}$ (let it be $r_i^{\bar{\eta}}$); we show that it also holds for $\underline{A}_{i+1}$. There is an ID that generates $\underline{A}_{i+1}$ from $\underline{A}_i$. By inductive hypothesis, since we are within the first $\delta_M$ levels, there must be a rule in $\Pi^{DC}$ corresponding to the ID in question, with an atom with predicate $r_i^{\bar{\eta}}$ in the body. The application of the immediate consequence operator on that rule will produce, by construction, an atom whose predicate annotation matches $\underline{A}_{i+1}$'s predicate and terms, and whose constants match $\underline{A}_{i+1}$'s innermost constants.

If $p$ is $eq$, the proof is as above, but instead of a single path, there may be multiple paths of the form $\underline{A}_0,\dots,\underline{A}_m = \underline{A}$; the above argument can be applied to any of them. The only difference is that, instead of ID rules, $eq$ atoms are generated either by KD rules in $\Pi^{\Sigma_K}$ or by the equality rules in $\Pi^{eq}$. For all such rules (and for all the atoms they are applied to) there are the corresponding annotated counterparts in $\Pi^{q,\Sigma}$ that have been added by the algorithm for rule annotation.

This proves that, apart from the $q_{eq}$ atoms, all the atoms in the first $\delta_M$ levels of $M_f$ have a corresponding annotated atom in $M$. Now, the algorithm for rule annotation has added to $\Pi^{q,\Sigma}$ all possible versions of $q_{eq}$ in which the head is annotated $q_{eq}^{\bullet,\dots,\bullet}$ and the positions in which the same variable occurs in the query are annotated in the same way, with all possible annotations occurring in the first $\delta_M$ levels of $M_f$. Therefore the $q_{eq}$ tuples in $M_f$ are contained in the $q_{eq}^{\bullet,\dots,\bullet}$ tuples in $M$.

For the other inclusion, we simply need to dispose the atoms in $M$ according to levels, as we did for the atoms in $M_f$. Starting from the atoms of $D$ in $M$ and the $eq$ atoms on constants in $D$, by the base annotation rules we obtain the same atoms with annotation $^{\bullet,\dots,\bullet}$; these annotated atoms are at level 0 in $M$; the nonannotated atoms are never used by any other rule in $\Pi^{q,\Sigma}$ and can be disregarded. Every other rule in $\Pi^{q,\Sigma}$, when used by the immediate consequence operator, generates an atom (in the head) starting from other atoms (in the body); when the generated atom is new, we draw an arc from each body atom to the head atom, and give it the level $\ell + 1$, where $\ell$ is the maximum level of the body atoms. The resulting structure is again a directed acyclic graph, and from this we can proceed as for the other inclusion and prove that for each atom in $M$, a corresponding nonannotated atom exists in $M_f$, since every rule produced by the algorithm for rule annotation, apart from $\Pi^{ba}$, is a syntactic variant of rules in $q_{eq} \cup \Pi^{eq} \cup \Pi^{\Sigma_K}$, and the rules in $\Pi^{DC}$ mimic the rules in $\Pi^{\Sigma_I}$. □

The above theorem suggests our final strategy for computing the answers to a conjunctive query $q$ expressed over an EER schema, given a database $D$.

(1) We derive a set $\Sigma$ of CDs that represent the EER schema.
(2) We check whether $chase_{\Sigma}(D)$ exists, as described in the proof of Lemma 1, in time polynomial in $|D|$.
(3) Then, we derive a Datalog rewriting that computes all certain answers to $q$, according to Theorem 4.
(4) Finally, we evaluate the Datalog rewriting on $D$.

### *5.3 Considerations on complexity*

We focus here on *data complexity*, i.e., the complexity w.r.t. the size of the data, that is the most relevant, since the size of the data is usually much larger than that of the schema.

*Proposition 2*
The complexity of computing the certain answers to a CQ over an EER schema is polynomial in the size of the data if the size $\lambda_D$ of the largest connected part in the join graph of the instance of the EER schema is bounded.

*Proof*
From a CQ $q$ over an EER schema, given a database $D$, we can proceed as follows.

(1) We check whether the chase exists, which can be done in polynomial time in the size of $D$ by Lemma 1; if it does not, then query answering is trivial (all $n$-tuples are in the answer to the query $q$, where $n$ is the arity of $q$); (2) we construct a Datalog rewriting for $q$, according to what was explained in the previous pages, which does not depend on $D$ but only on $\lambda_D$, which is assumed to be bounded; (3) we evaluate the rewriting on the data. Since the evaluation of a Datalog program is polynomial in data complexity (Dantsin *et al.* 2001), the thesis follows. □

### *5.4 Extensions of results*

**Dealing with inconsistencies.** First of all, as we mentioned in Section 4.2, we have always assumed that the initial, incomplete database satisfies the KDs derived from the EER schema. This assumption does not limit the applicability of our results, since violations of KDs can be treated in different ways. (1) *Data cleaning* (see, e.g., Hernández and Stolfo 1998): a preliminary cleaning procedure would eliminate the KD violations; then, the results from (Calì 2006) ensure that no violations will occur in the chase, and we can proceed with the techniques presented in the paper. (2) *Strictly sound semantics*: according to the sound semantics we have adopted, from the logical point of view, strictly speaking, a single KD violation in the initial data makes query answering trivial (any tuple is in the answer, provided it has the same arity of the query); this extreme assumption, not very usable in practice, can be encoded in suitable rules, which make use of inequalities, and that can be added to our rewritings. We refer the reader to (Calì *et al.* 2003b) for the details. (3) *Loosely-sound semantics*: this assumption is a relaxation of the previous one, and is reasonable in practice. Inconsistencies are treated in a model-theoretic way, and suitable Datalog$^{\neg}$ rules (that we can add to our programs without any trouble, obtaining a correct rewriting under this semantics) encode the reasoning on the constraints. Again, we refer the reader to (Calì *et al.* 2003b) for further details.

**Adding disjointness.** Disjointness between two classes, which is a natural addition to our EER model, can be easily encoded by *exclusion dependencies (EDs)* (see, e.g. Lembo 2004). The addition of EDs to CDs is not problematic, provided that we preliminarily compute the closure, w.r.t. the implication, of KDs and EDs, according to the (sound and complete) implication rules that are found in Lembo (2004). After that, we can proceed as in the absence of EDs.

## 6 Discussion

**Summary of results.** In this paper we have employed a conceptual model based on an extension of the ER model, which we called Extended Entity-Relationship, and we have given its semantics in terms of the relational database model with integrity constraints. We have thus carved out a relevant class of relational constraints, which is a subclass of the well-known key and IDs; such a class is important, because in real-world database design the constraints are directly derived from an ER schema. In fact, the focus of our contribution is on querying incomplete data under an interesting class of relational constraints, rather than on proposing another query language for EER schemata. Moreover, we argue that our results are independent of the translation from EER to relational.

We have considered conjunctive queries expressed over EER conceptual schemata, and we have tackled the problem of providing the certain answers to queries in such a setting, when the data are incomplete w.r.t. the constraints that encode the conceptual schema. We have characterized a class of relational constraints, namely CDs, which are able to represent EER schemata. This class is a subclass of KDs and IDs (in the general case the query answering problem is undecidable Calì *et al.* 2003a). In this way, we have reduced the query answering problem under EER constraints into the equivalent problem of query answering under CDs.

We have provided a query rewriting algorithm that transforms a conjunctive query $q$ into a new (recursive Datalog) query that, once evaluated on the incomplete data, returns the certain answers to $q$.

Finally, we have shown how our results can be extended to more general settings, in particular: (1) EER schema with class disjointness; (2) the so-called loosely-sound semantics for incomplete data, which overcomes the limitations of the strictly sound one.

**Related work.** Several works propose query languages for different flavors of EER schemata (Hohenstein and Engels 1992; Grant *et al.* 1993; Lawley and Topor 1994; Thalheim 2000). Our query language, which does not introduce novel features or characteristics, relies on a standard translation of EER schemata into relational ones.

https://doi.org/10.1017/S1471068410000104 Published online by Cambridge University Press

As pointed out earlier, query answering in our setting is tightly related to containment of queries under constraints, which is a fundamental topic in database theory (Johnson and Klug 1984; Chan 1992; Calvanese *et al.* 1998; Kolaitis and Vardi 1998). (Calì *et al.* 2001) deals with conceptual schemata in the context of data integration, but the cardinality constraints are more restricted than in our approach, since they do not include functional participation constraints and is-a among relationships.

Other works that deal with dependencies similar to those presented here are (Calvanese *et al.* 2005; Calvanese *et al.* 2006), which deal with a formalism called *DL-Lite* and based on Description Logic; it is easy to establish a correspondence between EER entities and DL-lite *concepts*, and between EER relationships and DL-lite (binary) *roles*. However, the set of constraints considered in the above works is not comparable to CDs: while it contains some constructs not expressible in EER, on the other hand it is unable to represent, for instance, the is-a among relationships, which we believe is the major source of complexity in the query answering problem. Also (Ortiz *et al.* 2006) addresses the problem of query containment using a formalism for the schema that is more expressive than the one presented here; the problem is proved to be coNP-hard. In Calvanese *et al.* (1998), the authors address the problem of query containment for queries on schemata expressed in a formalism that is able to capture our EER model; in this work it is shown that checking containment is decidable and its complexity is exponential in the number of variables and constants of $q_1$ and $q_2$, and doubly exponential in the number of existentially quantified variables that appear in a cycle of the *tuple-graph* of $q_2$ (we refer the reader to the paper for further details). Since the complexity is studied by encoding the problem in a different logic, it is not possible to analyze in detail the complexity w.r.t. $|q_1|$ and $|q_2|$, which by the technique of Calvanese *et al.* (1998) is in general exponential. If we export the results of Calvanese *et al.* (1998) to our setting, we get an exponential complexity w.r.t. the size of the data for the decision problem[4] of answering queries over incomplete databases. In our work we provide a technique that also serves the purpose of computing all answers to a query in the presence of incomplete data.

Our technique for dealing with the nonrepairable violations in the chase is the same as in Calì *et al.* (2003a). This is along the lines of consistent query answering (Arenas *et al.* 1999); a similar approach is found in Chomicki and Marcinkowski (2005).

**Future work.** As future work, we plan to extend the EER model with more constraints which are used in real-world cases, such as covering constraints or more sophisticated cardinality constraints. We also plan to further investigate the complexity of query answering, providing a thorough study of complexity, including lower complexity bounds. Also, we are working on an implementation of the query rewriting algorithm, so as to test the efficiency of our technique on large data sets.

[4] The decision problem of query answering amounts to deciding whether, given a query $q$ and a tuple $t$, $t$ belongs to the answers to $q$.

## Acknowledgments

Andrea Calì was supported by the EPSRC project EP/E010865/1 "Schema Mappings and Automated Services for Data Integration and Exchange." Davide Martinenghi was supported by the "Search Computing" (SeCo) project, funded by the ERC under the 2008 Call for "IDEAS Advanced Grants."

## References

Abiteboul, S., Hull, R. and Vianu, V. 1995. *Foundations of Databases*. Addison Wesley Publishing Co.

Arenas, M., Bertossi, L. E. and Chomicki, J. 1999. Consistent query answers in inconsistent databases. In *Proc. of PODS'99*, 68–79.

Bertossi, L. E. and Bravo, L. 2005. Consistent query answers in virtual data integration systems. In *Inconsistency Tolerance*, L. E. Bertossi, A. Hunter and T. Schaub, Eds. Lecture Notes in Computer Science, vol. 3300. Springer, 42–83.

Calì, A. 2003. Query answering and optimisation in information integration. PhD thesis, Universit di Roma "La Sapienza."

Calì, A. 2006. Containment of conjunctive queries over conceptual schemata. In *Proc. of DASFAA 2006*, 270–284.

Calì, A. 2007. Querying incomplete data with logic programs: Er strikes back. In *ER*, C. Parent, K.-D. Schewe, V. C. Storey and B. Thalheim, Eds. Lecture Notes in Computer Science, vol. 4801. Springer, 245–260.

Calì, A., Calvanese, D., De Giacomo, G. and Lenzerini, M. 2001. Accessing data integration systems through conceptual schemas. In *Proc. of ER 2001*, 270–284.

Calì, A., Calvanese, D., De Giacomo, G. and Lenzerini, M. 2004. Data integration under integrity constraints. *Information Systems 29*, 147–163.

Calì, A., Gottlob, G. and Kifer, M. 2008. Taming the infinite chase: Query answering under expressive relational constraints. In *Proc. of the 12th International Conference on the Principles of Knowledge Representation and Reasoning (KR 2008)*, 70–80.

Calì, A., Lembo, D. and Rosati, R. 2003a. On the decidability and complexity of query answering over inconsistent and incomplete databases. In *Proc. of PODS 2003*, 260–271.

Calì, A., Lembo, D. and Rosati, R. 2003b. Query rewriting and answering under constraints in data integration systems. In *Proc. of IJCAI 2003*, 16–21.

Calvanese, D., De Giacomo, G., Lembo, D., Lenzerini, M. and Rosati, R. 2005. DL-Lite: Tractable description logics for ontologies. In *Proc. of AAAI 2005*, 602–607.

Calvanese, D., De Giacomo, G., Lembo, D., Lenzerini, M. and Rosati, R. 2006. Data complexity of query answering in description logics. In *Proc. of the 10th International Conference on the Principles of Knowledge Representation and Reasoning (KR 2006)*, 260–270.

Calvanese, D., De Giacomo, G. and Lenzerini, M. 1998. On the decidability of query containment under constraints. In *Proc. of PODS'98*, 149–158.

Chan, E. P. F. 1992. Containment and minimization of positive conjunctive queries in OODB's. In *Proc. of PODS'92*, 202–211.

Chen, P. 1976. The Entity-Relationship model: Toward a unified view of data. *ACM Trans. on Database Systems 1,* 1 (March), 9–36.

Chomicki, J. and Marcinkowski, J. 2005. Minimal-change integrity maintenance using tuple deletions. *Information and Computation 197,* 1–2, 90–121.

Codd, E. F. 1970. A relational model of data for large shared data banks. *Communications of the ACM 13,* 6, 377–387.

Dantsin, E., Eiter, T., Gottlob, G. and Voronkov, A. 2001. Complexity and expressive power of logic programming. *ACM Computing Surveys 33,* 3, 374–425.

Deutsch, A., Nash, A. and Remmel, J. B. 2008. The chase revisited. In *PODS*, 149–158.

Duschka, O. M. and Genesereth, M. R. 1997. Answering recursive queries using views. In *Proc. of PODS'97*, 109–116.

Duschka, O. M. and Levy, A. Y. 1997. Recursive plans for information gathering. In *Proc. of IJCAI'97*, 778–784.

Fagin, R., Kolaitis, P. G., Miller, R. J. and Popa, L. 2005. Data exchange: semantics and query answering. *Theoretical Computer Science 336,* 1, 89–124.

Gottlob, G. and Nash, A. 2008. Efficient core computation in data exchange. *Journal of the ACM 55,* 2.

Grant, J., Ling, T. and Lee, M. 1993. ERL: Logic for entity-relationship databases. *Journal of Intelligent Information Systems 2,* 2, 115–147.

Hernández, M. A. and Stolfo, S. J. 1998. Real-world data is dirty: Data cleansing and the merge/purge problem. *J. of Data Mining and Knowledge Discovery 2,* 1, 9–37.

Hohenstein, U. and Engels, G. 1992. SQL/EER – syntax and semantics of an entity-relationship-based query language. *Information Systems 17,* 3, 209–242.

Johnson, D. S. and Klug, A. C. 1984. Testing containment of conjunctive queries under functional and inclusion dependencies. *Journal of Computer and System Sciences 28,* 1, 167–189.

Kolaitis, P. G. and Vardi, M. Y. 1998. Conjunctive-query containment and constraint satisfaction. In *Proc. of PODS'98*, 205–213.

Lawley, M. and Topor, R. 1994. A query language for EER schemas. In *Proceedings of ADC 1994*, 292–304.

Lembo, D. 2004. Dealing with inconsistency and incompleteness in data integration. PhD thesis, Dip. di Inf. e Sist., Univ. di Roma "La Sapienza."

Lenzerini, M. 2002. Data integration: A theoretical perspective. In *Proc. of PODS 2002*, 233–246.

Lloyd, J. W. 1987. *Foundations of Logic Programming (Second, Extended Edition)*. Springer, Berlin.

Maier, D., Mendelzon, A. O. and Sagiv, Y. 1979. Testing implications of data dependencies. *ACM Transactions on Database Systems 4*, 455–469.

Maier, D., Sagiv, Y. and Yannakakis, M. 1981. On the complexity of testing implications of functional and join dependencies. *Journal of the ACM 28,* 4, 680–695.

Markowitz, V. M. and Makowsky, J. A. 1990. Identifying extended entity-relationship object structures in relational schemas. *IEEE Transactions on Software Engineering 16,* 8, 777–790.

Ortiz, M., Calvanese, D. and Eiter, T. 2006. Characterizing data complexity for conjunctive query answering in expressive description logics. In *Proc. of the 21st National Conference on Artificial Intelligence (AAAI 2006)*, 275–280.

Reiter, R. 1978. On closed world data bases. In *Logic and Databases*, H. Gallaire and J. Minker, Eds. Plenum Publishing Co., 119–140.

Thalheim, B. 2000. *Entity Relationship Modeling – Foundations of Database Technology*. Springer.

Vardi, M. 1983. Inferring multivalued dependencies from functional and join dependencies. *Acta Informatica 19*, 305–324.