A Framework for Combining Entity Resolution and Query Answering in Knowledge Bases

Ronald Fagin\textsuperscript{1}, Phokion G. Kolaitis\textsuperscript{1,2}, Domenico Lembo\textsuperscript{3}, Lucian Popa\textsuperscript{1}, Federico Scafoglieri\textsuperscript{3}

\textsuperscript{1}IBM Research, Almaden, USA
\textsuperscript{2}UC Santa Cruz, USA
\textsuperscript{3}Sapienza University of Rome, Italy

fagin@us.ibm.com, kolaitis@ucsc.edu, lembo@diag.uniroma1.it, lpopa@us.ibm.com, scafoglieri@diag.uniroma1.it

Abstract

We propose a new framework for combining entity resolution and query answering in knowledge bases (KBs) with tuple-generating dependencies (tgds) and equality-generating dependencies (egds) as rules. We define the semantics of the KB in terms of special instances that involve equivalence classes of entities and sets of values. Intuitively, the former collect all entities denoting the same real-world object, while the latter collect all alternative values for an attribute. This approach allows us to both resolve entities and bypass possible inconsistencies in the data. We then design a chase procedure that is tailored to this new framework and has the feature that it never fails; moreover, when the chase procedure terminates, it produces a universal solution, which in turn can be used to obtain the certain answers to conjunctive queries. We finally discuss challenges arising when the chase does not terminate.

1 Introduction

Entity resolution is the problem of determining whether different data records refer to the same real-world object, such as the same individual or the same organization, and so on (Benjelloun et al. 2009; Papadakis et al. 2021). In this paper, we study entity resolution in combination with query answering in the context of knowledge bases (KBs) consisting of ground atoms and rules specified as tuple-generating dependencies (tgds) and equality-generating dependencies (egds). These rules have been widely investigated in databases and knowledge representation, e.g., in (Beeri and Vardi 1984; Fagin et al. 2005; Baget et al. 2011; Cuenca Grau et al. 2013; Krötzsch, Marx, and Rudolph 2019); in particular, they can express axioms that are used in Description Logic (DL) (Baader et al. 2007), as well as in specifying ontologies and KBs that are similar to Datalog +/- programs (Calì, Gottlob, and Lukasiewicz 2012). In addition, egds are employed to express typical entity resolution rules that one may write in practice, i.e., rules that enforce equality between two entities, as in (Bienvenu, Cima, and Gutiérrez-Basulto 2022).

The KBs considered here involve \( n \)-ary predicates that denote \( n \)-ary relations, \( n \geq 1 \), over entities and/or values from predefined datatypes. As is customary for ontologies, the TBox is the intensional component (i.e., the rules) of a KB, while the database of the KB is its extensional component (i.e., the ground atoms), sometimes called ABox.

As an example, Figure 1a depicts a set of ground atoms, where Doe_1 and Doe_2 are entities while the rest are values (indicating names and landline home phone numbers). Figure 2 illustrates a small TBox (containing only egds, for simplicity). To capture entity resolution rules, we allow egds to contain atoms involving built-in predicates, such as \texttt{JaccSim}. In the example, rule \( s_1 \) states that two names (i.e., strings) with Jaccard similarity above 0.6 must belong to the same individual. Rules \( s_2 \) and \( s_3 \) stipulate that an individual has at most one name and at most one landline home phone number, respectively. We call entity-egds rules that impose equality on two entities (e.g., \( s_1 \)), and we call value-egds rules that impose equality on two values (e.g., \( s_2 \) and \( s_3 \)).

It is now easy to see that, according to the standard semantics, the database in Figure 1a does not satisfy the entity-egd \( s_1 \) in Figure 2 (note that the Jaccard similarity of \texttt{John Doe} and \texttt{J. Doe} is 0.625). Thus, the main challenge one has in practice is to come up with a consistent way to complete or modify the original KB, while respecting all its rules.

In this paper, we develop a framework for entity resolution and query answering in KBs, where the valid models, called KB solutions, must satisfy all entity resolution rules along with all other KB rules; furthermore, the solutions must include all original data (i.e., no information is ever dropped or altered). Our approach is guided by the intuitive principle that for each real-world object there must be a single node in the solution that represents all “equivalent” entities denoting the object. To achieve this, we use equivalence classes of entities, which become first-class citizens in our framework. In addition, we relax the standard way in which value-egds are satisfied by allowing solutions to use sets of values, thus collecting together all possible values for a given attribute (e.g., the second argument of Name or HPhone). Continuing with the above example, Figure 1b shows a new set of ground atoms that uses equivalence classes and sets of values.

We remark that the use of equivalence classes of entities and sets of values in KB solutions requires a drastic revision of the classical notion of satisfaction for tgds and egds. Intuitively, we interpret egds as matching dependencies (Bertossi, Kolahi, and Lakshmanan 2013; Fan 2008). That is, when the conditions expressed by (the body of) an entity-egd or a value-egd hold in the data (and thus two en-
We propose a new framework for entity resolution in knowledge bases consisting of tgds and egds, and give rigorous semantics.

We define universal solutions and show that, as for standard tgd and egd semantics, universal solutions can be used to obtain the certain answers of CQs.

We propose a variant of the classical chase procedure (Beeri and Vardi 1984; Fagin et al. 2005) tailored to our approach. An important feature of our chase procedure is that it never fails, even in the presence of egds. At the same time, as in other frameworks, our chase procedure might not terminate.

We show that, when the chase procedure terminates, it returns a universal solution (and thus we have an algorithm for computing the certain answers to conjunctive queries).

However, by virtue of rule $s_2$, a solution has to group together the two telephone numbers into the same set, thus saying that Doe$_1$ and Doe$_2$ have both the landline home phone numbers $\{635, 358\}$. This union “fires” rule $s_1$, and thus entities are resolved, i.e., are put together into the same equivalence class of the solution.

We finally note that, in order to maximize entity resolution, our semantics allows the body of a rule to be satisfied by assignments in which different occurrences of the same variable are replaced by different sets of values, as long as these sets have a non-empty intersection (instead of requiring that all occurrences are replaced by the same set of values). For instance, if we add to rules in Figure 2 the tgd

$$HPhone(p_1, f) \land HPhone(p_2, f) \rightarrowSameHouse(p_1, p_2, f),$$

stating that two entities with the same home phone number live in the same house, and to the database in Figure 1a the atom $HPhone(Doe_3, 358)$, we can conclude that the individual denoted by Doe$_3$ and the individual denoted by Doe$_1$ and Doe$_2$ live in the same house (since they share one phone number). Consistently with this choice, we also require tgds to “propagate” such intersections. Then, through the above tgd, we also infer that the phone number of the house is $\{358\}$, which we denote with a fact of the form $SameHouse([Doe_1, Doe_2], [Doe_1, [358]]).$ The behaviour we described differs from the standard one only for variables ranging on values, since two equivalence classes of entities are always either disjoint or the same class.

In this paper, we formalize the aforementioned ideas and investigate query answering, with focus on conjunctive queries (CQs). The main contributions are as follows.

- We propose a new framework for entity resolution in knowledge bases consisting of tgds and egds, and give rigorous semantics.

- We define universal solutions and show that, as for standard tgd and egd semantics, universal solutions can be used to obtain the certain answers of CQs.

- We propose a variant of the classical chase procedure (Beeri and Vardi 1984; Fagin et al. 2005) tailored to our approach. An important feature of our chase procedure is that it never fails, even in the presence of egds. At the same time, as in other frameworks, our chase procedure might not terminate.

- We show that, when the chase procedure terminates, it returns a universal solution (and thus we have an algorithm for computing the certain answers to conjunctive queries).
Detailed proofs are given in the arXiv version of this article: https://arxiv.org/abs/2303.07469.

2 Basic Notions

We take for granted the notions of equivalence relation and equivalence class. If $Z$ is a set, $\theta$ is an equivalence relation on $Z,$ and $x \in Z$, then the equivalence class of $x$ w.r.t. $\theta$ is denoted by $[x]_\theta$ (or simply $[x]$, if $\theta$ is clear from the context). Sometimes, we write, e.g., $[a,b,c]_\theta$ (or $[a,b,c]$) to denote the equivalence class consisting of the elements $a, b,$ and $c$. The quotient set $Z/\theta$ of $\theta$ on $Z$ is the set of all equivalence classes over $Z$ w.r.t. $\theta$.

We consider four pairwise disjoint alphabets $S_P, S_E, S_V,$ and $S_Y$. The set $S_P$ is a finite alphabet for predicates; it is partitioned into the sets $S_O$ and $S_B$, which are the alphabets for KB predicates and built-in predicates, respectively. The sets $S_E, S_V,$ and $S_Y$ are countable infinite alphabets for entities, values, and variables, respectively. For ease of exposition, we do not distinguish between different data types, thus $S_Y$ is a single set containing all possible values.

The number of arguments of a predicate $P \in S_P$ is the arity of $P$, denoted with $arity(P)$. With each $n$-ary predicate $P$, we associate a tuple $type(P) = \langle \rho_1, \ldots, \rho_n \rangle$, such that, for each $1 \leq i \leq n$, either $\rho_i = e$ or $\rho_i = v$. This tuple specifies the types of the arguments of $P$, i.e., whether each argument of $P$ ranges over entities (e) or values (v). We also write $type(P, i)$ for the type $\rho_i$ of the $i$-th argument of $P$.

Note that the built-in predicates from $S_B$ are special, pre-interpreted predicates, whose arguments range only over values, i.e., if $B$ is an $n$-ary built-in predicate, then $type(B)$ is the $n$-ary tuple $(v, \ldots, v)$. The examples in Sec. 1 use the Jaccard similarity $JaccSim$ as a built-in predicate.

An atom is an expression $P(t_1, \ldots, t_n)$, where $P \in S_P$, and each $t_i$ is a term, i.e., is either a variable from $S_V$ or a constant, which in turn is either an entity from $S_E$, if $type(P, i) = e$, or a value from $S_Y$, if $type(P, i) = v$. When $t_i \in S_V$, we call it an entity-variable if $type(P, i) = e$, or a value-variable, if $type(P, i) = v$. A ground atom is an atom with no variables.

A conjunction $\phi(x)$ of atoms is an expression $P_1(t_1) \wedge \ldots \wedge P_m(t_m)$, where each $P_j(t_j)$ is an atom such that each variable in the tuple $t_j$ of terms is among those in the tuple $x$ of variables. We also require that every variable $x$ in $x$ is either an entity-variable or a value-variable and that, if $x$ occurs in an atom whose predicate is built-in (note that thus $x$ is a value-variable), then there exists some other atom $P_j(t_j)$ in $\phi(x)$ such that $P_j \in S_O$ and $x$ is in $t_j$. If a conjunction contains no built-in predicates, we say that it is built-in free; if it contains no entities or values, we say that it is constant-free.

As done in this section, in the formulas appearing throughout the paper, we use $e$ to denote an entity from $S_E$, we use $v$ to denote a value from $S_Y$, we use $c$ to denote a constant (i.e., $c \in S_E \cup S_Y$), we use $x, y, w, z$ to denote variables from $S_Y$, and we use $t$ to denote terms (i.e., $t \in S_Y \cup S_E \cup S_V$). Typically, we use $P$ to denote a predicate from $S_P$. All the above symbols may appear with subscripts. Moreover, we use bold font for tuples of variables, terms, and so on. In the examples, we use self-explanatory symbols; we write entities in Helvetica font and values in true type font.

3 Framework

In this section, we present the syntax and the semantics of a framework for entity resolution in knowledge bases.

Syntax. A knowledge base (KB) $K$ is a pair $(T, D)$, consisting of a TBox $T$ and a database $D$. The TBox is a finite set of tuple-generating dependencies (tgd) and equality-generating dependencies (egd). A tgd is a formula

$$\forall x(\phi(x) \rightarrow \exists y \psi(x, y)),$$

where $\phi(x)$ and $\psi(x, y)$ are conjunctions of atoms, such that $x$ and $y$ have no variables in common and $\psi(x, y)$ is built-in and, for simplicity, constant-free. As in (Fagin et al. 2005), we assume that all variables in $x$ appear in $\phi(x)$, but not necessarily in $\psi(x, y)$. We call $\phi(x)$ the body of the tgd, and $\psi(x, y)$ the head of it.

An egd is a formula

$$\forall x(\phi(x) \rightarrow y = z),$$

where $\phi(x)$ is a conjunction of atoms and $y$ and $z$ are distinct variables occurring in $\phi(x)$, such that either both $y$ and $z$ are entity-variables (in which case we have an entity-egd) or both $y$ and $z$ are value-variables (in which case we have a value-egd). We call $\phi(x)$ the body of the egd. For value-egd, we require that neither $y$ nor $z$ occur in atoms having a built-in as predicate. This ensures that the meaning of built-ins remains fixed. We will write $body(r)$ to denote the body of a tgd or an egd $r$; furthermore, we may write $r$ with no quantifiers.

Example 1. Let $T$ be the TBox consisting of the rules

$$(r_1)\ CI(p_1, name_1, phone_1) \land JaccSim(name_1, name_2, 0.6) \rightarrow p_1 = p_2$$
$$\quad (r_2)\ CI(p, name_1, phone_1) \land JaccSim(p, name_2, phone_2) \rightarrow name_1 = name_2$$
$$\quad (r_3)\ CI(p, name_1, phone_1) \land JaccSim(p, name_2, phone_2) \rightarrow phone_1 = phone_2$$
$$\quad (r_4)\ CI(p, name, phone) \rightarrow\ Emp(p, comp) \land CEO(comp, dir)$$
$$\quad (r_5)\ Emp(p, comp_1) \land Emp(p, comp_2) \rightarrow comp_1 = comp_2$$
$$\quad (r_6)\ CI(p_1, name_1, phone) \land CI(p_2, name_2, phone) \rightarrow SameHouse(p_1, p_2, phone).$$

Here, $type(CI) = \langle e, v, v \rangle$, $type(Emp) = \langle e, e \rangle$, $type(CEO) = \langle e, e \rangle$, and $type(SameHouse) = \langle e, e, v \rangle.$ The predicates have the meaning suggested by their names. In particular, the predicate $CI$ maintains contact information of individuals, whereas $Emp$ associates employees to the companies they work for. Each of the six rules makes an assertion about the predicates. In particular, rule $r_1$ states that if the Jaccard similarity of two names is higher then 0.6, then these names are names of the same individual.
The database $D$ of a KB $K$ is a finite set of ground atoms of the form $P(c_1, \ldots, c_n)$ over the alphabets $S_P$, $S_E$, and $S_V$, where $P \in S_P$ and each $c_i$ is an entity from $S_E$, if $type(P, i) = e$, or a value from $S_V$, if $type(P, i) = v$.

**Example 2.** Let $D$ be the database consisting of the atoms

$$(g_1)\ CI(Doe_1, J. Doe, 358), \ (g_2)\ CI(Doe_2, John Doe, 635), \ (g_3)\ Emp(Doe_1, Yahoo), \ (g_4)\ Emp(Doe_2, IBM), \ (g_5)\ CEO(Yahoo, Doe_1).$$

In words, the database $D$ specifies that $Doe_1$ has name $J. Doe$ and phone number $358$ ($g_1$), $Doe_2$ has name $John Doe$ and phone number $635$ ($g_2$), $Doe_3$ has name Mary Doe and phone number $358$ ($g_3$), $Doe_2$ is employee of Yahoo ($g_4$), $Doe_3$ is employee of IBM ($g_5$), and the CEO of Yahoo is $Doe_1$ ($g_6$).

To ensure that built-in predicates have the same semantics in every KB, we assume that we have a fixed (infinite and countable) set $GB$ of ground atoms of the form $B(v_1, \ldots, v_n)$, where $B$ is in $S_B$ and $v_1, \ldots, v_n$ are in $S_V$. Intuitively, $GB$ contains all facts about built-in predicates that hold overall. Given a KB $K = (T, D)$, we assume that $D = D_O \cup D_B$, where $D_O$ contains only ground atoms with predicate from $S_O$ and $D_B$ is the (finite) set of all atoms in $GB$ whose built-in predicates and values are mentioned in $T$ and in $D_O$.

**Semantics.** Let $S_{EN}$ and $S_{VN}$ be two infinite, countable, disjoint sets that are also disjoint from the alphabets introduced in Sec. 2. We call $S_{EN}$ the set of entity-nulls and $S_{VN}$ the set of value-nulls; their union is referred to as the set of nulls. We use $\sigma(K)$ to denote the signature of a KB $K$, i.e., the set of symbols of $S_P$, $S_E$ and $S_V$ occurring in $K$.

The semantics of a KB is given using special databases, called KB instances, whose ground atoms have components that are either equivalence classes of entities and entity-nulls or non-empty sets of values and value-nulls.

**Definition 1.** Let $K$ be a KB, $S$ a subset of $(S_E \cap \sigma(K)) \cup S_{EN}$, and $\sim$ an equivalence relation on $S$. An instance $I$ for $K$ w.r.t. $\sim$ is a set of facts $P(T_1, \ldots, T_n)$ such that $P \in S_P \cap \sigma(K)$, arity($P$) = $n$, and, for each $1 \leq i \leq n$, we have that $T_i \neq \emptyset$ and either $T_i \in S/\sim$, if type($P, i$) = $e$, or $T_i \subseteq (S_V \cap \sigma(K)) \cup S_{VN}$, if type($P, i$) = $v$. The relation $\sim$ is called the equivalence relation associated with $I$.

To denote an equivalence class in $S/\sim$, we may use the symbol $E$; also, to denote a non-empty subset of $(S_V \cap \sigma(K)) \cup S_{VN}$, we may use the symbol $V$. Even though $E$ may contain nulls, we will often call $E$ simply equivalence class of entities. Similarly, $V$ will be simply called set of values. We will then use $T$ to denote a set that is either $E$ or $V$. All such symbols may occur with a subscript.

**Definition 2.** Let $K$ be a KB and let $I$ be an instance for $K$ w.r.t. an equivalence relation $\sim$.

The active domain of $I$, denoted by active($I$), is the set $\{T |$ there are $P(T_1, \ldots, T_n) \in I$ and $i \leq n$ with $T = T_i\}$.

We write $active(E,I)$ and $active(V,I)$ to denote the set of all equivalence classes of entities and the set of all sets of values contained in $active(I)$, respectively. Obviously, $active(I) = active(E,I) \cup active(V,I)$.

The underlying domain of $I$, denoted by $under(I)$, is the set $under_E(I) \cup under_V(I)$, where

$under_E(I) = \{e |$ there is $E \in active_E(I)$ and $e \in E\}$

and

$under_V(I) = \{v |$ there is $V \in active_V(I)$ and $v \in V\}$.

Note that an instance $I$ of $K$ w.r.t. $\sim$ is also an instance of $K$ w.r.t. the equivalence relation $\sim'$ induced on $under_E(I)$ by $\sim$. Therefore, in what follows, we will consider only instances w.r.t. equivalence relations over $under_E(I)$.

Furthermore, we may simply call $I$ an instance for $K$ and leave the equivalence relation associated with $I$ implicit.

**Example 3.** Let $K = (T, D)$ be a KB such that $T$ and $D$ are as in Example 1 and in Example 2, respectively. Then, consider the following set $I$ of facts:

$$(d_1)\ CI([Doe_1], \{J. Doe\}, \{358\})$$

$$(d_2)\ CI([Doe_2], \{John Doe\}, \{635\})$$

$$(d_3)\ CI([Doe_1], \{Mary Doe\}, \{358\})$$

$$(d_4)\ Emp([Doe_1], \{Yahoo\})$$

$$(d_5)\ Emp([Doe_2], \{IBM\})$$

$$(d_6)\ CEO([Yahoo], [Doe_1]).$$

$I$ is an instance for $K$ w.r.t. the identity relation over the set $S = \{Doe_1, Doe_2, Doe_3, Yahoo, IBM\}$. Further, let $e_1^I$ and $e_2^I$ be entity-nulls, and $V^I$ be an equivalence relation over $S \cup \{e_1^I, e_2^I\}$, such that $Doe_1 \sim Doe_2$, IBM $\sim ^V e_2^I$, and their symmetric versions are the only equivalence classes in $\sim'$ different from the identity. The following set $I'$ of facts is an instance for $K$ w.r.t. $\sim'$.

$CI([Doe_1, Doe_2], \{J. Doe, John Doe\}, \{358, 635\})$

$CI([Doe_1], \{Mary Doe\}, \{358\})$

Emp([Doe_1], [Doe_2], [Yahoo])

Emp([Doe_1], [IBM], [e_2^I])

CEO([Yahoo], [Doe_1, Doe_2])

CEO([IBM, e_2^I], [e_2^I])$

SameHouse([Doe_1, Doe_2], [Doe_1, Doe_2], \{358\})

SameHouse([Doe_1], [Doe_1, Doe_2], \{358\}).$

To define the notions of satisfaction of tgds and egds by an instance, we first introduce the notion of an assignment from a conjunction $\phi(x)$ of atoms to an instance $I$ of a KB $K$. To formalize this notion, we need a preliminary transformation $\tau$ of $\phi(x)$ that substitutes each occurrence of a value-variable in $\phi(x)$ with a fresh variable. We call such fresh variables set-variables and denote the result of the transformation $\tau(\phi(x))$. If $x$ is a value-variable in $x$, we write SetVar($x, \tau(\phi(x))$) to denote the set of fresh variables used in $\tau(\phi(x))$ to replace the occurrences of $x$ in $\phi(x)$. For example, if $\phi(x) = P_1(x, y, z) \land P_2(y, z) \land P_3(x, w)$, where type($P_1$) = $\{e, e, v\}$, type($P_2$) = $\{e, v\}$, and type($P_3$) = $\{e, v\}$, then $\tau(\phi(x)) = P_1(x, y, S_2^I) \land P_2(y, S_2^I) \land P_3(x, S_1^I)$, where $S_2^I$, $S_2^I$, $S_1^I$ are the fresh set-variables introduced by $\tau$. Also, SetVar($x, \tau(\phi(x))$) = $\{S_2^I, S_2^I\}$ and SetVar($x, \tau(\phi(x))$) = $\{S_1^I\}$. In what follows, if $x$ is a value-variable, then each set-variable in SetVar($x, \tau(\phi(x))$) will have $x$ as a superscript.

We are now ready to formally define assignments.

**Definition 3.** Let $\phi(x)$ be a conjunction of atoms and let $I$ be an instance for a KB $K$ w.r.t. $\sim$. An assignment from $\phi(x)$ to $I$ is a mapping $\mu$ from the variables and values in $\tau(\phi(x))$ to active($I$), defined as follows:
1. $\mu(x)$ is an equivalence class in $\text{active}_E(\mathcal{I})$, for every entity-variable $x$;
2. $\mu(v) = V$ such that $V \in \text{active}_V(\mathcal{I})$ and $v \in V$, for every value $v$;
3. $\mu(S)$ is a set of values in $\text{active}_V(\mathcal{I})$, for every set-variable $S$;
4. $\bigcap_{i=1}^k \mu(S_i^x) \neq \emptyset$, for every value-variable $x$ such that
   $\text{SetVar}(x, \tau(\phi(x))) = \{S_1^x, \ldots, S_k^x\}$;
5. $\mathcal{I}$ contains a fact of the form $P(\mu(x), \mu(S), [\vdots], \mu(v))$, for each atom $P(x, S, e, v)$ of $\tau(\phi(x))$, where $x$ is an entity-variable, $S$ is a set-variable, $e$ is an entity in $\mathcal{E}_E$ and $v$ is a value in $\mathcal{E}_V$ (the definition generalizes in the obvious way for atoms of different form).

In words, the above definition says that an assignment maps every entity-variable to an equivalence class of entities (Condition 1), every value to a set of values containing it (Condition 2), and every occurrence of a value-variable to a set of values (Condition 3), in such a way that multiple occurrences of the same value-variable are mapped to sets with a non-empty intersection (Condition 4). This captures the intuition that, since predicate arguments ranging over values are interpreted through sets of values, a join between such arguments holds when such sets have a non-empty intersection. Finally, Condition 5 states that $\tau(\phi(x))$ is “realized” in $\mathcal{I}$. Note that an assignment also maps values to sets of values (Condition 2). This allows an assignment to map atoms in $\phi(x)$ to facts in $\mathcal{I}$, since predicate arguments ranging over values are instantiated in $\mathcal{I}$ by sets of values.

If $\mathcal{P}(t)$ is an atom of $\phi(x)$, we write $\mu(\mathcal{P}(t))$ for the fact of $\mathcal{I}$ in Condition 5 and call it the $\mu$-image (in $\mathcal{I}$) of $\mathcal{P}(t)$. We write $\mu(\phi(x))$ for the set $\{\mu(\mathcal{P}(t)) \mid \mathcal{P}(t) \text{ occurs in } \phi(x)\}$, and call it the $\mu$-image (in $\mathcal{I}$) of $\phi(x)$.

Example 4. Consider the tgds $r_6$ of Example 1 and the instance $\mathcal{I}$ given in Example 3 and apply $\tau$ to the body of $r_6$ to obtain $CI(p_1, S_{\text{name}_1}, S_{\text{phone}}^1) \land CI(p_2, S_{\text{name}_2}, S_{\text{phone}}^2)$. Let $\mu$ be the following mapping:

$\mu(p_1) = [\text{Doe}_1, \text{Doe}_2]$, $\mu(S_{\text{name}_1}) = \{\text{John Doe}\}$,
$\mu(S_{\text{phone}}^1) = \{358, 635\}$, $\mu(p_2) = [\text{Doe}_3]$, $\mu(S_{\text{name}_2}) = \{\text{Mary Doe}\}$, $\mu(S_{\text{phone}}^2) = \{358\}$.

It is easy to see that $\mu$ is an assignment from the body of $r_6$ to $\mathcal{I}$ (note the non-empty intersection between $\mu(S_{\text{phone}}^1)$ and $\mu(S_{\text{phone}}^2)$). Let us now apply $\tau$ to the head of $r_6$ to obtain $\text{SameHouse}(p_1, p_2, R_{\text{phone}})$. The following mapping $\mu'$ is an assignment from $\text{SameHouse}(p_1, p_2, \text{phone})$ to $\mathcal{I}'$:

$\mu'(p_1) = [\text{Doe}_1, \text{Doe}_2]$, $\mu'(p_2) = [\text{Doe}_3]$, $\mu'(R_{\text{phone}}) = \{358\}$.

We are now ready to define the semantics of tgds and egds.

Definition 4. An instance $\mathcal{I}$ for a KB $\mathcal{K}$ satisfies:

- a tgd of the form (1), if for each assignment $\mu$ from $\phi(x)$ to $\mathcal{I}$ there is an assignment $\mu'$ from $\psi(x, y)$ to $\mathcal{I}$ such that, for each $x$ in $x$ occurring in both $\phi(x)$ and $\psi(x, y)$:
  - $\mu(x) = \mu'(x)$, if $x$ is an entity-variable;
  - $\bigcap_{i=1}^n \mu(S_i^y) \subseteq \bigcap_{i=1}^m \mu'(R_i^y)$, where $\{S_1^y, \ldots, S_n^y\} = \text{SetVar}(x, \tau(\phi(x)))$ and $\{R_1^y, \ldots, R_m^y\} = \text{SetVar}(x, \tau(\psi(x, y)))$, if $y$ is a value-variable.

Each such assignment $\mu'$ is called a head-compatible tgd-extension of $\mu$ to $\mathcal{I}$ (or simply a tgd-extension of $\mu$ to $\mathcal{I}$). It is easy to see that $\mathcal{I}$ is not a solution for $\mathcal{K}$, whereas $\mathcal{I}'$ is a solution for $\mathcal{K}$.
Universal solutions. It is well known that the universal solutions exhibit good properties that make them to be the preferred solutions (see, e.g., (Fagin et al. 2005; Cali, Gottlob, and Kifer 2013; Calvanese et al. 2007)). To introduce the notion of a universal solution in our framework, we first need to adapt the notion of homomorphism. We begin with some auxiliary definitions and notation.

Definition 6. Let $T = ⟨T_1, . . . , T_n⟩$ and $T' = ⟨T'_1, . . . , T'_n⟩$ be two tuples such that each $T_i$ and each $T'_i$ is either a set of entities and entity-nulls or a set of values and value-nulls. 

- $T'$ dominates $T$, denoted $T \leq T'$, if $T_i \subseteq T'_i$, for all $i$.
- $T'$ strictly dominates $T$, denoted $T < T'$, if $T \leq T'$ and $T \neq T'$.
- Let $P(T)$ and $P(T')$ be facts. $P(T')$ dominates $P(T)$, denoted $P(T) \leq P(T')$, if $T \leq T'$; $P(T')$ strictly dominates $P(T)$, denoted $P(T) < P(T')$, if $T < T'$.

Definition 7. Let $I_1$ and $I_2$ be two instances of a KB $K$. A homomorphism $h : I_1 \rightarrow I_2$ is a mapping from the elements of $under(I_1)$ to elements of $under(I_2)$ such that:

1. $h(e) = e$, for every entity $e$ in $under_E(I_1) \cap S_E$;
2. $h(e_\perp)$ belongs to $under_E(I_2)$, for every entity-null $e_\perp$ in $under_E(I_1) \cap S_{EN}$;
3. $h(v) = v$, for every value $v$ in $under_V(I_1) \cap S_V$;
4. $h(v_\perp)$ belongs to $under_V(I_2)$, for every value-null $v_\perp$ in $under_V(I_1) \cap S_{V\perp}$;
5. for every $P(T_1, . . . , T_n)$ in $I_1$, there is a $P(U_1, . . . , U_n)$ in $I_2$ such that $P(h(T_1), . . . , h(T_n)) \leq P(U_1, . . . , U_n)$, where $h(T_i) = \{h(x) \mid x \in T_i\}$, for $1 \leq i \leq n$.

In the sequel, we may use $h(⟨T_1, . . . , T_n⟩)$ to denote the tuple $⟨h(T_1), . . . , h(T_n)⟩$.

We now define the key notion of a universal solution.

Definition 8. A solution $U$ for a KB $K$ is universal if, for every $I \in Sol(K)$, there is a homomorphism $h : U \rightarrow I$.

The instance $I'$ in Example 3 is a universal solution for $K$, as is the instance obtained by eliminating $\Theta^+_1$ from $I'$. Two instances $I_1$ and $I_2$ are homomorphically equivalent if there are homomorphisms $h : I_1 \rightarrow I_2$ and $h' : I_2 \rightarrow I_1$. All universal solutions are homomorphically equivalent.

4 Query Answering

A conjunctive query (CQ) $q$ is a formula $\exists y \phi(x, y)$, where $\phi(x, y)$ is a built-in free conjunction of atoms. The arity of $q$ is the number of its free variables in $x$; we will often write $q(x)$, instead of just $q$, to indicate the free variables of $q$.

Let $q(x)$ : $\exists y \phi(x, y)$ be a CQ, where $x = x_1, . . . , x_n$. Given a KB $K$ and an instance $I$ for $K$, the answer to $q$ on $I$, denoted by $q^I$, is the set of all tuples $⟨T_1, . . . , T_n⟩$ such that there is an assignment $\mu$ from $\phi(x, y)$ to $I$ for which

- $T_i = \mu(x_i)$, if $x_i$ is an entity-variable;
- $T_i = \bigcap_{k=1}^n \mu(S_{\tau_i}^x)$, if $x_i$ is a value-variable, where $\{S_{\tau_i}^x, . . . , S_{\tau_k}^x\} = SetVar(x_i, \tau(\phi(x, y)))$.

We will also say that $\mu$ is an assignment from $q(x)$ to $I$.

Example 7. Let $q(x)$ be the CQ:

$$\exists p_1, p_2 CI(p_1, J. Doe, x) \land CI(p_2, Mary Doe, x)$$

asking for the phone number in common between (the entities named) $J. Doe$ and Mary Doe. The answer to $q(x)$ on the instance $I'$ in Example 3 is $q^I' = \{(358)\}$. This is obtained through the assignment $\mu_4$ defined as follows:

- $\mu_4(p_1) = \{\text{Doe1}, \text{Doe2}\}$, $\mu_4(J. Doe) = \{J. Doe, John Doe\}$
- $\mu_4(S_1) = \{358, 635\}$, $\mu_4(p_2) = \{\text{Doe3}\}$
- $\mu_4(Mary Doe) = \{\text{Mary Doe}, \mu_4(S_2) = \{358\}$.

If $q_1(x) : \exists z CEO(z, x)$ is the query asking for the CEOs, then $q_1^I = \{\{(\text{Doe1}, \text{Doe2})\}, \{(\text{Doe2})\}\}$.

The next result tells how conjunctive queries are preserved under homomorphisms in our framework.

Proposition 1. Let $q$ be a CQ and let $K$ be a KB. If $I_1, I_2$ are two instances for $K$ and $h : I_1 \rightarrow I_2$ is a homomorphism from $I_1$ to $I_2$, then, for every $T \in q^I_1$, there is $U \in q^I_2$ such that $h(T) \subseteq U$.

When querying a KB $K$, we are interested in reasoning over all solutions for $K$. We adapt the classical notion of certain answers to our framework. A tuple $T = ⟨T_1, . . . , T_n⟩$ is null-free if each $T_i$ is non-empty and contains no nulls.

Definition 9. Let $q$ be a CQ and let $K$ be a KB. A null-free tuple $T$ is a certain answer to $q$ w.r.t. $K$ if

1. for every solution $I$ for $K$, there is a tuple $T' \in q^I$ such that $T \leq T'$;
2. there is no null-free tuple $T'$ that satisfies 1. and $T < T'$.

We write $cert(q, K)$ for the set of certain answers to $q$.

Note that the second condition in the above definition asserts that a certain answer has to be a maximal null-free tuple with respect to the tuple dominance order $\leq$ given in Definition 6, among the tuples satisfying Condition 1.

The next result tells that if two sets of entities appear in a certain answer or in different certain answers, then either they are the same set or they are disjoint. In particular, the sets of entities that appear in certain answers can be viewed as equivalence classes of some equivalence relation.

Proposition 2. Let $q$ be a CQ of arity $n$, let $K$ be a KB, and let $T = ⟨T_1, . . . , T_n⟩$ and $T' = ⟨T'_1, . . . , T'_n⟩$ be two certain answers to $q$ w.r.t. $K$. If $T_i$ and $T'_i$ are sets of entities, then either $T_i = T'_i$ or $T_i \cap T'_i = \emptyset$, where $1 \leq i, j \leq n$.

In data exchange and related areas, the certain answers to CQs can be obtained by evaluating the query on a universal solution and then applying an operator $\bot$ that eliminates the tuples that contain nulls (see, e.g., (Fagin et al. 2005; Calvanese et al. 2007)). The operator $\bot$ can easily be adapted to our framework as follows. If $E$ is a set of entities and entity-nulls, then $E_\bot = \{e \mid e \in S_E \cap E\}$. Similarly, if $V$ is a non-empty set of values and value-nulls, then $V_\bot = \{v \mid v \in S_V \cap V\}$. In words, $\bot$ removes all nulls from $E$ and from $V$. If $⟨T_1, . . . , T_n⟩$ is a tuple such that each $T_i$ is either a set of entities and entity-nulls or a set of values and value-nulls, then we set $⟨T_1, . . . , T_n⟩_\bot = ⟨T_1\bot, . . . , T_n\bot⟩$. Finally, given a set $\Theta$ of tuples of the above form, $\Theta_\bot$ is the
set obtained from $\Theta$ by removing all tuples in $\Theta$ containing a $T_i$ such that $T_i \downarrow = \emptyset$, and replacing every other tuple $(T_1, \ldots, T_n) \downarrow$ in $\Theta$ by the tuple $(T_1, \ldots, T_n) \downarrow$.

The next example shows that, in our framework, the certain answers to a CQ $q$ cannot always be obtained by evaluating $q$ on a universal solution and then applying $\downarrow$.

**Example 8.** Let $P_1$ and $P_2$ be such that $\text{type}(P_1) = \text{type}(P_2) = \langle e, v \rangle$, $T'$ be the TBox consisting of the rules

$$P_1(x, y) \rightarrow P_2(x, y), \quad P_1(x, y) \land P_1(x, z) \rightarrow y = z$$

and $D' = \{ P_1(\langle e, 1 \rangle), P_1(\langle e, 2 \rangle) \}$. Consider the following universal solutions for $K' = (T', D')$:

$$I_1 = \{ P_1(\langle e, \{1, 2\} \rangle), P_2(\langle e, \{1, 2\} \rangle) \}, \quad I_2 = \{ P_1(\langle e, \{1, 2\} \rangle), P_2(\langle e, \{1\} \rangle), P_2(\langle e, \{1, 2\} \rangle) \},$$

and the query $q(x, y) = P_2(x, y)$. Then $q^{I_1} = q^{I_2} = \{ \langle e, \{1, 2\} \rangle \}$, and $q^{I_1} \downarrow = q^{I_2} \downarrow = \{ \langle e, \{1\} \rangle \}$. Thus, $q \downarrow$ does not coincide with the set of certain answers to $q$ with respect to $K'$, because the tuple $\langle e, \{1\} \rangle$ does not satisfy the second condition in Definition 9.

Intuitively, in the above example the universal solution $I_2$ is not “minimal”, in the sense that the fact $P_2(\langle e, \{1\} \rangle)$ in $I_2$ is dominated by another fact of $I_2$, namely $P_2(\langle e, \{1, 2\} \rangle)$. This behaviour causes that the answer to the query over $I_2$ contains also tuples that are not maximal with respect to tuple dominance (cf. Definition 9). This suggests that we need some additional processing besides elimination of nulls. We modify the operator $\downarrow$ by performing a further reduction step.

**Definition 10.** Given a query $q$ and an instance $I$ for a KB $K$, we write $q^{\downarrow}_I$ to denote the set of null-free tuples obtained by removing from $q^{\downarrow}_I$ all tuples strictly dominated by other tuples in the set, that is, if $T$ and $T'$ are two tuples in $q^{\downarrow}_I$ such that $T < T'$, then remove $T$ from $q^{\downarrow}_I$.

We are now able to present the main result of this section, which asserts that universal solutions can be used to compute the certain answers to CQs in our framework.

**Theorem 1.** Let $q$ be a CQ, let $K$ be a KB, and let $\mathcal{U}$ be a universal solution for $K$. Then $\text{cert}(q, K) = q^{\downarrow}_{\mathcal{U}}$.

**Example 9.** As seen earlier in Example 6, $I'$ is a universal solution for $K$. Then, for the query $q_1(x) = \exists z\text{CEO}(x, z)$, we have that $\text{cert}(q_1, K) = q^{\downarrow}_{I'} = \{ \langle \text{Doe}_1, \text{Doe}_2 \rangle \}$.

5 Computing a Universal Model

In this section, we adapt the well-known notion of restricted chase (Beeri and Vardi 1984; Johnson and Klug 1984; Fagin et al. 2005; Calvanese et al. 2007) to our framework. Interestingly, the chase procedure we define never produces a failure, unlike, e.g., the restricted chase procedure in the case of standard data exchange, where the application of egds may cause a failure when two different constants have to be made equal. Instead, in our framework, when an entity-egd forces two different equivalence classes of entities to be equated, we combine the two equivalence classes into a bigger equivalence class. Similarly, when a value-egd forces two different sets of values to be equated, we take the union of the two sets and modify the instance accordingly. However, it is possible that the chase procedure may have infinitely many steps, each producing a new instance. As a consequence, some care is required in defining the result of the application of this (potentially infinite) procedure to a KB $K$, so that we can obtain an instance that can be used (at least in principle) for query answering. We call such instance the result of the chase of the KB $K$, and distinguish the case in which the chase terminates from the case in which it does not. In the former case, the result of the chase of $K$ is simply the instance produced in the last step of the chase procedure, and we show that this instance is a universal solution for $K$. In the latter case, we point out that previous approaches from the literature for infinite standard chase sequences under tgds and egds cannot be smoothly adapted to our framework, and we leave it open for this case how to define the result of the chase so that it is a universal solution.

We start with the notion of the base instance for a KB. Given a KB $K = (T, D)$, we define the set

$$I^D = \{ P(\langle c_1 \rangle, \ldots, \langle c_n \rangle) \mid P(c_1, \ldots, c_n) \in D \}.$$  

$I^D$ is an instance for $K$ with respect to the identity relation id over the set $S_E \cap \text{sig}(K)$. We call $I^D$ the base instance for $K$. Note that the base instance for $K$ is also a base instance for every KB having $D$ as database, and is a solution for $(\emptyset, D)$. As an example, note that the instance $I$ of Example 3 is the base instance for the KB $K$ defined in Example 1 and in Example 2.

We next define three chase steps, one for tgds, one for entity-egds, and one for value-egds.

**Definition 11.** Let $K = (T, D)$ be a KB and $I_1$ an instance for $K$ with respect to the equivalent relation $\sim$ on under$_E(I_1)$.

- (tgds) Let $r$ be a tgd of the form (1). Without loss of generality, assume that all atoms in $\psi(x, y)$ are of the form $P(x_1, \ldots, x_k, y_1, \ldots, y_l)$, where $x_1, \ldots, x_k$ belong to $x$ and $y_1, \ldots, y_l$ belong to $y$, and we denote with $x = y_1, \ldots, y_k$ and $y = y_1, \ldots, y_l$ the entity-variables and value-variables in $\psi$, respectively. Let $\mu$ be an assignment from $\phi(x)$ to $I_1$ such that there is no $\psi(x, y)$-compatible tgd-extension of $\mu$ to $I_1$. We say that $r$ is $\text{applicable}$ to $I_1$ with $\mu$ (or that $\mu$ $\text{triggers}$ $r$ in $I_1$), and construct $I_2$ via the following procedure:

1. Let $\{ f_1^1, \ldots, f_k^1 \} \subseteq S_{EN}$ and $\{ f_1^l, \ldots, f_k^l \} \subseteq S_{VN}$ be two sets of fresh nulls (i.e., not occurring in $I_1$), which are distinct from each other.

2. Put $I_2 := I_1$

3. For each atom $P(x_1, \ldots, x_k, y_1, \ldots, y_l)$ in $\psi(x, y)$ do:

   $$I_2 := I_2 \cup \{ P(T_1, \ldots, T_k, U_1, \ldots, U_l) \},$$

where,

- for $1 \leq i \leq k$, we have $T_i = \mu(x_i)$, if $x_i$ is an entity-variable, or $T_i = \bigcap_{p=1}^m \mu(S_{SP}^{x_i})$, if $x_i$ is a value-variable, and $\{ S_{SP}^{x_1}, \ldots, S_{SP}^{x_k} \} = \text{SetVar}(x_i, \tau(\phi(x)))$;

- for $1 \leq i \leq \ell$, we have $U_i = f_i^1$ if $y_i = y_i^0$, with $1 \leq s \leq h$, or $U_i = f_i^j$ if $y_i = y_i^q$, with $1 \leq q \leq j$.

(Thus, each singleton $\{ f_i^s \}$ is a new equivalence class.)
• (entity-egd) Let \( r \) be an entity-egd of the form (2), and \( \mu \) an assignment from \( \phi(x) \) to \( \mathcal{I}_1 \) such that \( \mu(y) \neq \mu(z) \).

We say that \( r \) is applicable to \( \mathcal{I}_1 \) with \( \mu \) (or that \( r \) triggers \( \mathcal{I}_1 \)), and we construct \( \mathcal{I}_2 \) from \( \mathcal{I}_1 \) by replacing in \( \mathcal{I}_1 \) all occurrences of \( \mu(y) \) and \( \mu(z) \) with \( \mu(y) \cup \mu(z) \).

(Thus, we merge two equivalence classes into a new one.)

• (value-egd) let \( r \) be a value-egd of the form (2). Let \( \{S_1^y, \ldots, S_m^y\} = \text{SetVar}(y, \tau(\phi(x))) \), \( \{S_1^z, \ldots, S_k^z\} = \text{SetVar}(z, \tau(\phi(x))) \), \( 1 \leq i \leq m, 1 \leq j \leq k \), and \( \mu \) be an assignment from \( \phi(x) \) to \( \mathcal{I}_1 \) such that \( \mu(S_i^y) \neq \mu(S_j^z) \).

We say that \( r \) is applicable to \( \mathcal{I}_1 \) with \( \mu \) (or that \( r \) triggers \( \mathcal{I}_1 \)), and we construct \( \mathcal{I}_2 \) from \( \mathcal{I}_1 \) by replacing in the image \( \phi(x) \) each set \( \mu(S_i^y), \ldots, \mu(S_m^y), \mu(S_j^z), \ldots \mu(S_k^z) \) and \( \mu(S_i^y) \cup \ldots \mu(S_m^y) \cup \mu(S_j^z) \cup \ldots \cup \mu(S_k^z) \).

If \( r \) is a tgd or egd that can be applied to \( \mathcal{I}_1 \) with \( \mu \), we say that \( \mathcal{I}_2 \) is the result of applying \( r \) to \( \mathcal{I}_1 \) with \( \mu \) and we write \( \mathcal{I}_1 \xrightarrow{r, \mu} \mathcal{I}_2 \). We call \( \mathcal{I}_1 \xrightarrow{r, \mu} \mathcal{I}_2 \) a chase step.

For both the entity-egd step and the value-egd step, the chase procedure constructs \( \mathcal{I}_2 \) by replacing some facts of \( \mathcal{I}_1 \). However, whereas for entity-egds the replacement is “global” (i.e., the two equivalence classes merged in the step are substituted by their union everywhere in \( \mathcal{I}_1 \)), for value-egds the replacement is “local”, in the sense that the two sets merged in the step are substituted by their union only in facts occurring in the image \( \phi(x) \), which is a subset of \( \mathcal{I}_1 \).

Example 10. Consider again the KB \( \mathcal{K} = (\mathcal{T}, \mathcal{D}) \) of Example 1 and Example 2. The instance \( \mathcal{I} \) of Example 3 is the base instance \( \mathcal{I}^D \) for \( \mathcal{K} \). We depict below the application of the rules of Definition 11, starting from the instance \( \mathcal{I} = \mathcal{I}^D \).

**tgd application:** \( \mathcal{I}^D \xrightarrow{r_1, \mu_0} \mathcal{I}_1 \), where \( \mu_0 \) is such that \( \mu_0(\text{body}(r_4)) = \{d_3\} \). The set \( \mathcal{I}_1 \) consists of the facts \( d_1 \cdot d_6 \) (see Example 3), as well as the facts:

\[
(d_r) \text{Emp}([\text{John Doe}, \{e_1^+\}], (d_8) \text{CEO}([e_1^+], [e_2^+])).
\]

**entity-egd application:** \( \mathcal{I}_1 \xrightarrow{r_1, \mu_1} \mathcal{I}_2 \), where \( \mu_1 \) is such that \( \mu_1(\text{body}(r_1)) = \{d_1, d_2, \text{JaccSim}([\{\text{J. Doe}, \{\text{John Doe}\}, \{0, 6\}]])\} \). The set \( \mathcal{I}_2 \) consists of the facts \( d_5, d_6, d_7, d_8 \), as well as the facts:

\[
(d_1) \rightarrow (d_9) \text{Cl}([\text{Doe1}, \{\text{J. Doe}\}, \{358\}]),
(d_2) \rightarrow (d_{10}) \text{Cl}([\text{Doe2}, \{\text{John Doe}\}, \{0, 6\}]),
(d_4) \rightarrow (d_{11}) \text{Emp}([\text{Doe1}, \text{Doe2}, \{\text{Yahoo}\}]),
(d_6) \rightarrow (d_{12}) \text{CEO}([\text{Yahoo}], [\text{Doe1}, \text{Doe2}]).
\]

**value-egd application:** \( \mathcal{I}_2 \xrightarrow{r_2, \mu_2} \mathcal{I}_3 \), where \( \mu_2 \) is such that \( \mu_2(\text{body}(r_2)) = \{d_9, d_{10}\} \). The set \( \mathcal{I}_3 \) consists of the facts \( d_{13}, d_{14} \), as well as the facts:

\[
(d_9) \rightarrow (d_{13}) \text{Cl}([\text{Doe1}, \text{Doe2}, \{\text{J. Doe, John Doe}\}, \{358\}]),
(d_{10}) \rightarrow (d_{14}) \text{Cl}([\text{Doe1}, \text{Doe2}, \{\text{J. Doe, John Doe}\}, \{635\}]).
\]

We now define the notion of a chase sequence.
Lemma 1. Let $K$ be a KB and let $\sigma = \sigma_0, \sigma_1, \ldots, \sigma_m$ be a finite chase for $K$. Then $\text{chase}(K, \sigma)$ is a solution for $K$.

The following lemma is used to prove that if $\sigma$ is a finite chase, then $\text{chase}(K, \sigma)$ is not just a solution for $K$, but also a universal solution for $K$. A similar result, known as the Triangle Lemma, was used in (Fagin et al. 2005) in the context of data exchange.

Lemma 2. Let $I_1 \stackrel{r_\mu}{\rightarrow} I_2$ be a chase step. Let $I$ be an instance for $K$ such that $I$ satisfies $r$ and there exists a homomorphism $h_1 : I_1 \rightarrow I$. Then there is a homomorphism $h_2 : I_2 \rightarrow I$ such that $h_2$ extends $h_1$.

The following theorem is the main result of this section.

Theorem 2. If $K$ is a KB and $\sigma = \sigma_0, \sigma_1, \ldots, \sigma_m$ is a finite chase for $K$, then $\text{chase}(K, \sigma)$ is a universal solution for $K$.

Theorem 2 implies that if both $\sigma$ and $\sigma'$ are finite chases for $K$, then $\text{chase}(K, \sigma)$ and $\text{chase}(K, \sigma')$ are homomorphically equivalent. Thus, the result of a finite chase for $K$ is unique up to homomorphic equivalence.

It is easy to verify that if $K = (T, D)$ is a KB in which every tgd in $T$ is full (i.e., there are no existential quantifiers in the heads of the tgd in $T$), then every chase sequence for $K$ is finite. In particular, this holds true if $T$ consists of egds only, which covers all entity resolution settings.

Infinite chase sequences. For infinite chase sequences, the definition of the result of the chase requires some care. The typical approach adopted when only tgd are present (Krotzsch, Marx, and Rudolph 2019; Grahne and Onet 2018), or for settings involving separable tgd and egd (Johnson and Klug 1984; Calvanese et al. 2007; Cali, Gottlob, and Kifer 2013), is to define the result of an infinite chase sequence as the union of all facts generated in the various chase steps. This approach does not work for arbitrary tgd and egd since the resulting instance needs not satisfy all the rules and thus needs not be a solution in our terminology. Moreover, in our framework the union of all instances in an infinite chase sequence is not even an instance for the KB at hand, because the sets of entities in the result do not correspond to equivalence classes with respect to an equivalence relation.

An alternative approach, which might be better suited for a setting with arbitrary tgd and egd, is to define the result of an infinite chase sequence as the instance containing all persistent facts, i.e., all facts that are introduced in some step in the chase sequence and are never modified in subsequent chase steps. This notion was introduced in (Beerli and Vardi 1984). In our setting, given a KB $K$ and an infinite chase sequence $\sigma = I_0, I_1, \ldots$, this means that we define the result $\text{chase}(K, \sigma)$ of the infinite chase $\sigma$ of $K$ as follows:

$$\text{chase}(K, \sigma) = \{ f \mid \text{there is some } i \geq 0 \text{ such that } f \in I_j \text{ for each } j \geq i \}.$$  (3)

The following example shows that this definition does not work in our framework, because the above set might be empty (even if the database $D$ is non-empty).

Example 12. Let $K = (T, D)$, where $D = \{ P(1, 2) \}$ and $T$ consists of the two rules:

$$(r_1) \quad P(x, y) \rightarrow P(y, z)$$

$$(r_2) \quad P(x, y) \land P(y, z) \rightarrow y = z$$

with $\text{type}(P) = \langle v, v \rangle$. We construct an infinite chase sequence starting with $I_0 = \{ P(1, 2) \}$ and by repeatedly applying the above rules with suitable assignments in the following order: $r_1, r_2, r_1, r_2, r_1, r_2, r_1, r_2, \ldots$.

We obtain the infinite chase sequence $\sigma = I_0, I_1, \ldots$,

$$I_0 = \{ P(1, 2) \}$$

$$I_1 = \{ P(1, 2), P(2, v_1) \}$$

$$I_2 = \{ P(1, 2), P(2, v_1), P(3, v_2) \}$$

$$I_3 = \{ P(1, 2, v_1), P(2, v_1), P(3, v_1), P(3, v_1), P(3, v_1), P(3, v_1), P(3, v_1), P(3, v_1) \}$$

$$\ldots$$

$$I_7 = \{ P(1, 2, v_1, v_2), P(2, v_1, v_2), P(3, v_1, v_2), P(3, v_1, v_2), P(3, v_1, v_2), P(3, v_1, v_2), P(3, v_1, v_2) \}$$

$$\ldots$$

It is not difficult to see that the set $\text{chase}(K, \sigma)$ defined in (3) above is empty, i.e., no persistent facts occur in $\sigma$.  

The infinite chase sequence $\sigma$ in Example 12 is far. At the same time, there are finite fair sequences for the KB $K$ in this example. Indeed, if we apply rule $r_1$ and then rule $r_2$, we get a finite chase sequence $\sigma' = I_0, I_1', I_2'$, where

$$I_0' = \{ P(1, 2) \}$$

$$I_1' = \{ P(1, 2), P(2, v_1) \}$$

$$I_2' = \{ P(1, 2, v_1), P(2, v_1), P(3, v_1) \}.$$
Consequently, $\text{chase}(\mathcal{K}, \sigma') = T_2$. Thus, further investigation is needed when both finite and infinite chase sequences exist for a KB. We leave this as a topic for future work.

6 Related Work

From the extensive literature on entity resolution, and due to space limitations, we comment briefly on only a small subset of earlier work that is related to ours.

Swoosh (Benjelloun et al. 2009) is a generic approach to entity resolution in which the functions used to compare and merge records are “black boxes”. In our framework, the match function is determined by entity-egds and value-egds, whereas the merge function is implemented via the union operation - an important special case of merge functions in the Swoosh approach. In this sense, our framework is less general than Swoosh. In another sense, our framework is more general as it supports tgds, differentiates between entities and values, and incorporates query answering.

As in the works on matching dependencies (MDs) (Fan 2008; Bertossi, Kolahi, and Lakshmanan 2013; Bahmani et al. 2012), we consider variables in the head of egds to be matched and merged rather than to be made equal. In (Bertossi, Kolahi, and Lakshmanan 2013), generic functions obeying some natural properties are used to compute the result of a match, while we use the union of the sets of values or sets of entities involved in the match. In the MDs framework, a version of the chase procedure is used to resolve dependency violations. This chase acts locally on a pair of tuples at each step, whereas our chase matches entities in a global fashion and values in a local fashion. Furthermore, we support tgds, which are not in the framework of MDs. There is a body of work on declarative entity resolution and its variants that is related to our framework: relevant references include the Dedupalog framework (Arasu, Ré, and Suciu 2009), the declarative framework for entity linking in (Burdick et al. 2016; Burdick et al. 2019), and the more recent LACE framework (Bienvenu, Cima, and Gutiérrez-Basulto 2022). In both Dedupalog and LACE, there is a distinction between hard and soft entity resolution rules. LACE supports global merges and creates equivalence classes as we do, but it does not support local merges. A more recent extension, called LACE+ (Bienvenu et al. 2023) (which was not yet published at the time of our work), combines both global merges and local merges, as in our framework. Some important differences are that LACE+ combines entity resolution rules with denial constraints, while our framework combines entity resolution rules with tgds. Furthermore, they consider the complexity of various problems such as the existence of solutions and deciding whether a merge is certain or not, while we focus on the chase and on the query answering problem. The declarative framework for entity linking in (Burdick et al. 2016; Burdick et al. 2019) uses key dependencies and disjunctive constraints with “weighted” semantics that measure the strength of the links. The key dependencies are interpreted as hard rules that the solutions must satisfy in the standard sense. Consequently, two conflicting links cannot co-exist in the same solution, hence that approach uses repairs to define the notion of the certain links. In contrast, in our framework we carry along via the chase all the alternatives as either sets of values or equivalence classes of entities. Another feature of our framework is the focus on the certain answers of queries. Finally, the aforementioned declarative framework for entity linking uses more general link relations without rules for an equivalence relation of entities; thus, it is less focused on entity resolution.

7 Conclusions and Future Work

The main contribution of this paper is the development of a new declarative a framework that combines entity resolution and query answering in KBs. This is largely a conceptual contribution because the development of the new declarative framework entailed rethinking from first principles the definitions of such central notions as assignment, homomorphism, satisfaction of tgds and egds in a model, and universal solution. At the technical level, we designed a chase procedure that never fails, and showed that, when it terminates, it produces a universal solution that, in turn, can be used in query answering.

As regards future directions, perhaps the most pressing issue is to identify a “good” notion of the result of the chase when the chase procedure does not terminate. This may lead to extending the framework presented here to settings where all universal solutions are infinite. While infinite universal solutions cannot be materialized, such solutions have been used to obtain rewritability results (Calvanese et al. 2007), occasionally combined with partial materialization of the result of an infinite chase (Lutz, Toman, and Wolter 2009). In parallel, it is important to identify structural conditions on the tgds and egds of the TBox that guarantee termination of the chase procedure in polynomial time and, thus, yield tractable conjunctive query answering. It is also worthwhile enriching our framework with other kinds of axioms, e.g., denial constraints as in (Bienvenu, Cima, and Gutiérrez-Basulto 2022), and exploring whether other variants of the chase procedure, such as the semi-oblivious (a.k.a. Skolem) chase (Marnette 2009; Calautti and Pieris 2021) or the core chase (Deutsch, Nash, and Remmel 2008), can be suitably adapted to our framework so that their desirable properties carry over.

Finally, we note that there are several different areas, including data exchange (Fagin et al. 2005), data integration (Lenzerini 2002; Doan, Halevy, and Ives 2012), ontology-mediated query answering (Bienvenu and Ortiz 2015), and ontology-based data access (Calvanese et al. 2018), in which tgds and egds play a crucial role. We believe that the framework presented here makes it possible to infer entity resolution into these areas in a principled way.

Acknowledgments

Kolaitis’ research was partially supported by NSF Award No. 1814152. Lembo and Scafoglieri were supported by EU ICT-48 2020 project TAILOR (No. 952215), EU ERA-NET Cofund ICT-AGRI-FOOD project ADCATER (No. 40705), and PNRR MUR project PE0000013-FAIR.
References


