Ontologies for Knowledge Graphs: Breaking the Rules

Markus Krötzsch(✉) and Veronika Thost

Center for Advancing Electronics Dresden (cfaed), TU Dresden, Dresden, Germany
{markus.kroetzsch,veronika.thost}@tu-dresden.de

Abstract. Large-scale knowledge graphs (KGs) are widely used in industry and academia, and provide excellent use-cases for ontologies. We find, however, that popular ontology languages, such as OWL and Datalog, cannot express even the most basic relationships on the normalised data format of KGs. Existential rules are more powerful, but may make reasoning undecidable. Normalising them to suit KGs often also destroys syntactic restrictions that ensure decidability and low complexity. We study this issue for several classes of existential rules and derive new syntactic criteria to recognise well-behaved rule-based ontologies over KGs.

1 Introduction

Graph-based representations are playing a major role in modern knowledge management. Their simple, highly normalised data models can accommodate a huge variety of different information sources, and led to large-scale *knowledge graphs* (KGs) in industry (e.g., at Google and Facebook); on the Web (e.g., Freebase [6] and Wikidata [26]); and in research (e.g., YAGO2 [16] and Bio2RDF [5]).

Not all data is graph-shaped, but it is usually easy to translate into this format using well-known methods. For example, the W3C *RDB to RDF Mapping Language* provides mappings from relational databases to RDF graphs [13]. Relational tuples with three or more values are represented by introducing new graph nodes, to which the individual values of the tuple are then connected directly. For example, the tuple spouse(ann, jo, 2013), stating that Ann married Jo in 2013, may be represented by the graph in Fig. 1, where c is a fresh element introduced for this tuple, and s_1 to s_3 are binary edge labels used for all tuples of the spouse relation.

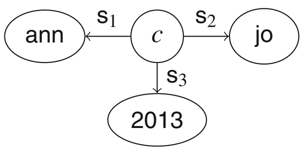

Fig. 1. Tuple as Graph

In this way, KGs unify data formats, so that many heterogeneous datasets can be managed in a single system. Unfortunately, however, syntactic alignment is not the same as semantic integration. The KG's flexibility and lack of schematic constraints lead to conceptual heterogeneity, which reduces the KG's utility. This is a traditional data integration problem, and ontologies promise to solve it in an interoperable and declarative fashion [19]. Indeed, ontologies can be used to model semantic relationships between different structures, so that a coherent global view can be obtained.

P. Groth et al. (Eds.): ISWC 2016, Part I, LNCS 9981, pp. 376–392, 2016.
DOI: 10.1007/978-3-319-46523-4_23

It therefore comes as a surprise that ontologies are so rarely used with KGs. A closer look reveals why: modern ontology languages cannot express even the simplest relationships on KG models. In our example, a natural relationship to model would be that marriage is symmetric, so that we can infer $\mathsf{spouse}(\mathsf{jo}, \mathsf{ann}, \mathsf{2013})$. In a KG, this fact would again be represented by a structure as in Fig. 1, but with Ann and Jo switched, and – importantly – using a fresh auxiliary node in place of c. This entailment could be expressed by the following logical axiom:

$$\forall x, y_1, y_2, y_3.\ \mathsf{s}_1(x, y_1) \wedge \mathsf{s}_2(x, y_2) \wedge \mathsf{s}_3(x, y_3) \ \rightarrow\ \exists v.\ \mathsf{s}_1(v, y_2) \wedge \mathsf{s}_2(v, y_1) \wedge \mathsf{s}_3(v, y_3). \quad (1)$$

Two ontology languages proposed for information integration in databases are *global-as-view* and *local-as-view* mappings [19]. Neither can express (1), since they support only single atoms on the source and on the target side, respectively. *Datalog*, a popular language for defining recursive views, cannot express (1) either, since it lacks existential quantification in conclusions of rules. Another very popular ontology language is OWL [22], which was specifically built for use with RDF graphs. However, even OWL cannot express (1): it supports rules with existential quantifiers, but only with exactly one universally quantified variable occurring in both premise and conclusion.

This problem is not specific to our particular example. KGs occur in many formats, which are rarely as simple as RDF. It is, e.g., common to associate additional information with edges. Examples are validity times in YAGO2, statement qualifiers in Wikidata, and arbitrary edge attributes in Property Graphs (a very popular data model in graph databases). If we want to represent such data in a simple relational form that is compatible with first-order predicate logic, we arrive at encodings as in Fig. 1.

So how can we realise ontology-based information integration on KGs? Formula (1) is in fact what is called a *tuple-generating dependency* in databases [1] and an *existential rule* in AI [2]. While query answering over such rules is undecidable, many decidable fragments have been proposed (see overviews [2], [8], and [11]). These rules use a relational model, and they can be translated to a KG setting just like facts. For example, rule (1) could be the result of translating $\forall y_1, y_2, y_3.\mathsf{spouse}(y_1, y_2, y_3) \rightarrow \mathsf{spouse}(y_2, y_1, y_3)$. However, this changes the rules' syntax and semantics, and it destroys known criteria that guarantee decidability or complexity.

We therefore ask to which extent known decidable fragments of existential rules are applicable to KGs, and we propose alternative definitions where necessary, to recover desirable properties. Our main results are:

– We show that *acyclicity* criteria and related complexities are generally preserved when transforming rules to KGs, and we identify a restricted class of acyclic rules that comprises transformed Datalog and retains its complexity.
– We show that the transformation destroys other basic syntactic criteria such as *linearity* and *guardedness*, though it preserves the underlying semantic notions (FO-rewritability and tree-like model property).
– We propose a new way of *denormalising* KG rules, based on the intuition that several edges can be grouped into "objects", and we exhibit cases for which this approach succeeds in producing rule sets that fall into known decidable classes.
– We introduce a notion of *incidental functional dependency*, which we use to extend our denormalisation to wider classes of rules, and we exhibit a sound procedure for computing such dependencies.

In all cases, we develop criteria that significantly generalise the motivating scenario of translating relational ontologies to KGs. In practice, it is more realistic to assume that ontologies are constructed over KGs directly. In this case, one cannot expect rules to have a regular structure as obtained by a rigid syntactic transformation, but patterns guaranteeing decidability and complexity bounds might still be identifiable.

Full proofs are available in an extended technical report [18].

2 Preliminaries

We briefly introduce essential notation and define the important notion of *graph normalisation*. We consider a standard language of first-order predicate logic, using *predicates* p of *arity* $\mathsf{ar}(p)$, *variables*, and *constants*. A *term* is a constant or variable. Finite lists of variables etc. are denoted in bold, e.g., $\boldsymbol{x}$. We use the standard predicate logic definitions of *atom* and *formula*. An *existential rule* (or simply *rule*) is a formula of form $\forall \boldsymbol{x}, \boldsymbol{y}.\varphi[\boldsymbol{x}, \boldsymbol{y}] \to \exists \boldsymbol{v}.\psi[\boldsymbol{x}, \boldsymbol{v}]$ where φ and ψ are conjunctions of atoms, called the *body* and *head* of the rule, respectively. Rules without existentially qualified variables are *Datalog rules*. We usually omit the universal quantifiers when writing rules.

We separate input relations (EDB) from derived relations (IDB). Formally, for a set of rules $\mathbb{P}$, the predicate symbols that occur in the head of some rule are called *intensional* (or *IDB*); other predicates are called *extensional* (or *EDB*). A *fact* is an atom that contains no variables. A *database* $\mathbb{D}$ is a set of facts over EDB predicates. A *conjunctive query* (CQ) is a formula $\exists \boldsymbol{y}.\varphi[\boldsymbol{x}, \boldsymbol{y}]$, where φ is a conjunction of atoms. A *Boolean CQ* (BCQ) is a CQ without free variables.

We only consider rules without constants. They can be simulated as usual, by replacing every constant a in a rule by a new variable x_a, adding the atom $O_a(x_a)$ to the body, and extending the database to include a single fact $O_a(a)$.

Rules and databases can be evaluated under a first-order logic semantics, and we use $\models$ to denote the usual first-order entailment relation between (sets) of formulae. CQ answering over existential rules can be reduced to BCQ entailment, i.e., the problem of deciding if $\mathbb{D}, \mathbb{P} \models \exists \boldsymbol{y}.\varphi$ holds for a given BCQ $\exists \boldsymbol{y}.\varphi$, database $\mathbb{D}$, and set of rules $\mathbb{P}$ [1]. This is undecidable in general, but many special classes of rule sets have been identified where decidability is recovered; we will see several examples later.

We now formalise the standard transformation of n-ary facts into directed graphs that was given in the introduction, and extend it to rules over n-ary predicates.

Definition 1. *For every predicate p, let $p_1, \ldots, p_{\mathsf{ar}(p)}$ be fresh binary predicates. Given an atom $p(\boldsymbol{t})$ and a term s, the* graph normalisation $\mathsf{GN}(s, p(\boldsymbol{t}))$ *is the set* $\{p_1(s, t_1), \ldots, p_{\mathsf{ar}(p)}(s, t_n)\}$ *of binary atoms. For a database $\mathbb{D}$, define* $\mathsf{GN}(\mathbb{D})$ *to be the union of the sets* $\mathsf{GN}(c_A, A)$ *for all facts $A \in \mathbb{D}$ where c_A is a fresh constant for A. For a rule $\rho = B_1 \wedge \ldots \wedge B_m \to \exists \boldsymbol{v}.H_1 \wedge \ldots \wedge H_\ell$, let* $\mathsf{GN}(\rho)$ *be the rule* $\bigwedge_{i=1}^{m} \mathsf{GN}(z_i, B_i) \to \exists \boldsymbol{v}.\exists \boldsymbol{w}. \bigwedge_{j=1}^{\ell} \mathsf{GN}(w_j, H_j)$ *using fresh variables $\boldsymbol{z}$ and $\boldsymbol{w}$. For a set of rules $\mathbb{P}$, let* $\mathsf{GN}(\mathbb{P}) := \bigcup_{\rho \in \mathbb{P}} \mathsf{GN}(\rho)$.

Example 1. Consider a database about PhD graduates and theses with facts of the form $\mathsf{sup}(\mathit{person}, \mathit{supervisor})$ and $\mathsf{phd}(\mathit{person}, \mathit{thesis\ title}, \mathit{date})$. We can express that every supervisor of a PhD graduate also has a PhD, using P for inferred (IDB) PhD relations:

$$\mathsf{phd}(x, y_1, y_2) \to \mathsf{P}(x, y_1, y_2) \tag{2}$$

$$\mathsf{P}(x_1, y_1, y_2) \wedge \mathsf{sup}(x_1, x_2) \to \exists v_1, v_2.\mathsf{P}(x_2, v_1, v_2) \tag{3}$$

The graph normalisation of this rule set is as follows:

$$\mathsf{phd}_1(z, x) \wedge \mathsf{phd}_2(z, y_1) \wedge \mathsf{phd}_3(z, y_2) \to \exists v.\mathsf{P}_1(v, x) \wedge \mathsf{P}_2(v, y_1) \wedge \mathsf{P}_3(v, y_2) \tag{4}$$

$$\begin{aligned} \mathsf{P}_1(z_1, x_1) \wedge \mathsf{P}_2(z_1, y_1) \wedge \mathsf{P}_3(z_1, y_2) \wedge \mathsf{sup}_1(z_2, x_1) \wedge \mathsf{sup}_2(z_2, x_2) \\ \to \exists v, v_1, v_2.\mathsf{P}_1(v, x_2) \wedge \mathsf{P}_2(v, v_1) \wedge \mathsf{P}_3(v, v_2) \end{aligned} \tag{5}$$

3 Acyclicity

Sets of existential rules may require models to be infinite. An immediate approach for ensuring decidability is to consider criteria that guarantee the existence of a finite universal model, which can be fully computed and used to answer queries. This led to many so-called *acyclicity* criteria [11]. We review one of the simplest cases, *weak acyclicity*.

Definition 2. *A* position *in a predicate p is a pair $\langle p, i\rangle$, where $i \in \{1, \ldots, \mathsf{ar}(p)\}$. The* dependency graph *G of a rule set $\mathbb{P}$ is defined as follows. The vertices of G are all positions of predicates in $\mathbb{P}$. For every rule $\varphi[\boldsymbol{x}, \boldsymbol{y}] \to \exists \boldsymbol{v}.\psi[\boldsymbol{x}, \boldsymbol{v}] \in \mathbb{P}$: (1) G has an edge from $\langle p, i\rangle$ to $\langle q, j\rangle$ if $x \in \boldsymbol{x}$ occurs at position $\langle p, i\rangle$ in φ and at $\langle q, j\rangle$ in ψ; (2) G has a* special *edge from $\langle p, i\rangle$ to $\langle q, j\rangle$ if $x \in \boldsymbol{x}$ occurs at position $\langle p, i\rangle$ in φ and there is an existentially quantified variable $v \in \boldsymbol{v}$ at $\langle q, j\rangle$ in ψ.*

$\mathbb{P}$ is weakly acyclic *if its dependency graph does not contain a directed cycle that involves a special edge.*

Theorem 1. *If $\mathbb{P}$ is weakly acyclic, then so is $\mathsf{GN}(\mathbb{P})$. Analogous preservation properties hold for rule sets that are jointly acyclic, super-weakly acyclic, model-faithful acyclic, or that have an acyclic graph of rule dependencies.*

While most acyclicity notions are thus preserved, this is not a general rule: *model-summarising acyclicity* (MSA) might be destroyed by graph normalisation [18].

BCQ entailment for acyclic rule sets is 2ExpTime-complete [11]. Datalog, however, enjoys a lower ExpTime-complete complexity [12], so Theorem 1 does not yield tight complexity estimates there. ExpTime complexity bounds for acyclic rules were given for rule sets where the maximal length of paths in a (slightly different) type of dependency graph is bounded [17, Theorem 5]. This condition is implied by the following property:

Theorem 2. *If $\mathbb{P}$ is a set of Datalog rules, then the dependency graph of $\mathsf{GN}(\mathbb{P})$ is such that every path contains at most one special edge.*

The number of special edges on paths can therefore be used to recognise (generalisations of) graph-normalised Datalog for which CQ answering is in ExpTime.

4 Beyond Acyclicity

Acyclicity is only one of several approaches for determining that reasoning is decidable for a given set of existential rules. It turns out, however, that other syntactic criteria are not as robust when applying graph normalisation to a set of rules, although one can show that essential semantic properties are preserved.

Baget et al. have identified several general classes of rule sets for which reasoning is decidable [2]. Acyclic rule sets are a typical form of *finite expansion set* (fes), which have a finite universal model. Rule sets without this property may still have an infinite universal model that is sufficiently "regular" to be presented finitely. This is the case if there is a universal model of bounded treewidth, leading to *bounded treewidth sets* (bts). A third general class of practical importance are *finite unification sets* (fus), corresponding to the class of first-order rewritable rule sets for which conjunctive queries (CQs) can be rewritten into finite unions of CQs (UCQs).

All of these abstract properties are preserved during graph normalisation. For fes and bts, this can be shown by noting that any (universal) model of $\mathbb{P}$ can be transformed into a (universal) model of $\mathsf{GN}(\mathbb{P})$ by treating it like an (infinite) database and applying $\mathsf{GN}(\cdot)$. For fus, the result follows since we can apply graph normalisation to the UCQ rewriting to obtain a valid rewriting for $\mathsf{GN}(\mathbb{P})$.

Theorem 3. *If* $\mathbb{P}$ *is fes/bts/fus, then* $\mathsf{GN}(\mathbb{P})$ *is fes/bts/fus.*

However, membership in these abstract classes is undecidable, so we need simpler sufficient conditions in practice. We disregard fes here, since it is already covered in Sect. 3. For bts, an easy-to-check criterion is (frontier) *guardedness* [2]:

Definition 3. *A rule* $\varphi[\boldsymbol{x}, \boldsymbol{y}] \to \exists \boldsymbol{v}.\psi[\boldsymbol{x}, \boldsymbol{v}]$ *is* frontier guarded *if* φ *contains an atom that contains all variables of* $\boldsymbol{x}$*. A rule set* $\mathbb{P}$ *is frontier guarded if all of its rules are.*

Frontier guarded rule sets are bts, and, by Theorem 3, so are their graph normalisations. Unfortunately, this is not easy to recognise, since frontier guardedness is often destroyed when breaking apart body atoms during graph normalisation. For instance, the original rules in Example 1 are frontier guarded, but the normalised rule (4) is not. The only general criterion that could recognise bts in normalised rules is *greedy bts* [4]; but a procedure for recognising this criterion has not been proposed yet, and the problem is generally assumed to be of very high complexity.

The situation is similar for fus. One of the simplest syntactic conditions for this case is *linearity* (a.k.a. *atomic hypothesis* [2]):

Definition 4. *An existential rule is* linear *if its body consists of a single atom. A rule set* $\mathbb{P}$ *is linear if all of its rules are.*

Again, this condition is clearly not preserved by graph normalisation. For example, rule (2) is linear while rule (4) is not.

Towards a way of recognising fus and bts rules even after graph normalisation, we look for ways to undo this transformation, i.e., to *denormalise* the graph. A natural approach of reversing the transformation from $p(\boldsymbol{x})$ to $p_1(z, x_1) \wedge \ldots \wedge p_n(z, x_n)$ is to group atoms by their first variable z. We may think of such groups of atoms as *objects* (as in object-oriented programming), motivating the following terminology.

Definition 5. *Consider a rule $\varphi \to \exists \boldsymbol{v}.\psi$. An* object *in φ (or ψ) is a maximal conjunction of atoms of the form $p_1(z, x_1) \wedge \ldots \wedge p_n(z, x_n)$ that occur in φ (or ψ), where neither variables x_i nor predicates p_i need to be mutually distinct. We call z* object variable*, $p_1, \ldots, p_n$* attributes*, and $x_1, \ldots, x_n$* values *of the object. The* interface *of the object is the set of variables $\boldsymbol{y} \subseteq \{x_1, \ldots, x_n\}$ occurring in atoms in $\varphi \to \exists \boldsymbol{v}.\psi$ that do not belong to the object.*

Note that each object is confined to either body or head, but cannot span both. In general, several attributes of an object may share a value, and several objects may use the same attributes. The definition therefore generalises the specific conjunctions of binary attributes introduced in graph normalisation. Existential rules may be thought of as "creating" new objects when using existential object variables. It is suggestive to use objects for defining KG versions of the above criteria:

Definition 6. *A rule $\varphi[\boldsymbol{x}, \boldsymbol{y}] \to \exists \boldsymbol{v}.\psi[\boldsymbol{x}, \boldsymbol{v}]$ over binary predicates is* pseudo KG linear *if φ consists of a single object. It is* pseudo KG frontier guarded *if φ contains an object ξ where all variables of $\boldsymbol{x}$ occur in. A rule is* KG linear *(*KG frontier guarded*) if it is pseudo KG linear (pseudo KG frontier guarded), and no object variable occurs as a value in any object.*

The "pseudo" versions of the above notions are not enough to obtain the desired properties, as the following example illustrates.

Example 2. The following rules are pseudo KG frontier guarded:

$$\mathsf{p}(z, x) \to \mathsf{P}(z, x) \tag{6}$$

$$\mathsf{P}(z, x) \to \exists w_1, w_2.\mathsf{H}(z, w_1) \wedge \mathsf{V}(z, w_2) \tag{7}$$

$$\mathsf{H}(z, y_1) \wedge \mathsf{V}(z, y_2) \to \exists v, w.\mathsf{P}(v, w) \wedge \mathsf{H}(y_2, v) \wedge \mathsf{V}(y_1, v) \tag{8}$$

where p is EDB and the other predicates are IDB. However, the rules are not bts, since applying them to the database with fact $\mathsf{p}(a, b)$ leads to models in which V and H form (possibly among other things) an infinite grid – a structure of unbounded treewidth.

5 Graph Denormalisation

To understand how and when our intuition of "objects" can be used to recognise rules with good properties, we introduce a systematic process for *denormalising* rules. Its goal is to replace objects $p_1(z, x_1) \wedge \ldots \wedge p_n(z, x_n)$ by single atoms of the form $\mathsf{D}(z, \boldsymbol{x}')$, while preserving semantics. D is a new predicate for this specific object. Note that $\boldsymbol{x}'$ can be limited to the interface of the object with its rule. For example, rule (5) contains the object $\mathsf{P}_1(z_1, x_1) \wedge \mathsf{P}_2(z_1, y_1) \wedge \mathsf{P}_3(z_1, y_2)$, but y_1 and y_2 do not occur in any other object in body or head. One could therefore replace this object by $\mathsf{D}_\mathsf{P}(z_1, x_1)$, and add a *defining rule*

$$\mathsf{P}_1(z_1, x_1) \wedge \mathsf{P}_2(z_1, y_1) \wedge \mathsf{P}_3(z_1, y_2) \to \mathsf{D}_\mathsf{P}(z_1, x_1) \tag{9}$$

to preserve semantics. We do not need the reverse implication, since D is used in the body only. The defining rule is essential to ensure completeness, but it is still in a normalised syntactic form that is usually not acceptable. To address this, we eliminate defining rules by rewriting them using resolution ("backward chaining"). We define this here for the special case of rewriting defining rules for single objects:

Definition 7. *Consider rules $\rho_1 : \varphi_1 \wedge \bar{\varphi}_1 \to \mathsf{D}(z, \boldsymbol{x})$ where $\varphi_1 \wedge \bar{\varphi}_1$ is a single object, and $\rho_2 : \varphi_2 \to \exists \boldsymbol{v}.(\psi_2 \wedge \bar{\psi}_2) \wedge \xi$ where $\psi_2 \wedge \bar{\psi}_2$ is a single object, so that ρ_1 and ρ_2 do not share variables. If there is a substitution θ that maps variables of ρ_1 to variables of ρ_2 such that $\bar{\varphi}_1\theta = \bar{\psi}_2$, and $\varphi_1\theta$ does not contain any variables from $\boldsymbol{v}$, then the rule $\varphi_1\theta \wedge \varphi_2 \to \exists \boldsymbol{v}.\mathsf{D}(z, \boldsymbol{x})\theta \wedge \xi$ is a* rewriting of ρ_1 using ρ_2*. We also consider rewritings of rules that share variables, assuming that variables are renamed apart before rewriting.*

Notice that we do not require $\bar{\varphi}_1$ to be the maximal part of the body object for which a rewriting is possible, as is common in (Boolean) conjunctive query rewriting [2]. Doing so would be incomplete, since we need to derive all possible bindings for $\mathsf{D}(x, \boldsymbol{y})$, which may require different parts to be unified with different rule heads. On the other hand, it is sufficient for our purposes to weaken the result by omitting the remaining head object parts ψ_2.

Example 3. Rewriting rule (9) with rules (4) and (5) yields two rules

$$\mathsf{phd}_1(z, x) \wedge \mathsf{phd}_2(z, y_1) \wedge \mathsf{phd}_3(z, y_2) \to \exists v.\mathsf{D}_\mathsf{P}(v, x) \quad (10)$$

$$P_1(z_1, x_1) \wedge P_2(z_1, y_1) \wedge P_3(z_1, y_2) \wedge \mathsf{sup}_1(z_2, x_1) \wedge \mathsf{sup}_2(z_2, x_2) \to \exists v.\mathsf{D}_\mathsf{P}(v, x_2). \quad (11)$$

Since the P_i are IDB predicates that only follow from rules (4) and (5), this represents all possible ways to infer new information using rule (9), and we can omit the latter. The bodies of rules (10) and (11) can be denormalised by adding further auxiliary predicates:

$$\mathsf{D}_\mathsf{phd}(z, x, y_1, y_2) \to \exists v.\mathsf{D}_\mathsf{P}(v, x) \quad (12)$$

$$\mathsf{D}_\mathsf{P}(z_1, x_1) \wedge \mathsf{D}_\mathsf{sup}(z_2, x_1, x_2) \to \exists v.\mathsf{D}_\mathsf{P}(v, x_2) \quad (13)$$

where D_phd and D_sup are EDB predicates that need to be defined by denormalising the database, and D can be re-used. We have therefore found a way of expressing (9) in terms of denormalised rules.

Our basic denormalisation algorithm needs to rewrite defining rules exhaustively, and might require to rewrite the same rule several times using its own rewritings, with variables renamed to avoid clashes. For a rule ρ_1 and rule set $\mathbb{P}$, we therefore define `rewrite`$(\rho_1, \mathbb{P})$ to be the result (least fixed point) of the following recursive process:

– Initialise `rewrite`$(\rho_1, \mathbb{P}) := \mathbb{P}$.
– Add to `rewrite`$(\rho_1, \mathbb{P})$ every rewriting of ρ_1 using some rule in `rewrite`$(\rho_1, \mathbb{P})$.
– Repeat the previous step until no further changes occur.

This approach terminates and `rewrite`$(\rho_1, \mathbb{P})$ is finite since each new rewriting contains fewer head objects than the rule used to obtain it. In particular, only rules with more than a single head object may ever require multiple rewritings.[1]

[1] For existential rules, replacing $\varphi \to \psi_1 \wedge \psi_2$ by two rules $\varphi \to \psi_1$ and $\varphi \to \psi_2$ is only correct if ψ_1 and ψ_2 do not share existential variables. Rules with multiple head objects are therefore unavoidable in general. Inseparable parts of rule heads are called *pieces* [2].

Algorithm 1. Generic denormalisation algorithm

```
Input  : rule set $\mathbb{P}$; database $\mathbb{D}$
Output: denormalised rule set $\mathsf{Result}_{\mathbb{P}}$ and denormalised database $\mathsf{Result}_{\mathbb{D}}$
1  $\mathsf{Todo} := \{\varphi \to \mathsf{D}_{\varphi}(z, \boldsymbol{x}) \mid \varphi$ an object with object term $z$ and interface $\boldsymbol{x}$ in a rule body of $\mathbb{P}\}$
2  $\mathsf{Done} := \emptyset$
3  $\mathsf{Rules} := \mathbb{P}$
4  while there is some rule $\rho \in \mathsf{Todo}$ do
5      $\mathsf{Todo} := \mathsf{Todo} \setminus \{\rho\}$
6      $\mathsf{Done} := \mathsf{Done} \cup \{\rho\}$
7      foreach $(\varphi \to \exists \boldsymbol{v}.\psi) \in \mathtt{rewrite}(\rho, \mathsf{Rules})$ do
8          foreach body object $\xi[z, \boldsymbol{x}]$ with object term $z$ and interface $\boldsymbol{x}$ in $\varphi \to \exists \boldsymbol{v}.\psi$ do
9              if there is $\xi'[z', \boldsymbol{x}'] \to \mathsf{D}(z', \boldsymbol{x}') \in \mathsf{Done}$ such that $\xi[z, \boldsymbol{x}] \equiv \xi'[z', \boldsymbol{x}']$ then
10                 replace $\xi[z, \boldsymbol{x}]$ in $\varphi$ by $\xi'[z, \boldsymbol{x}]$
11             else
12                 $\mathsf{Todo} := \mathsf{Todo} \cup \{\xi[z, \boldsymbol{x}] \to \mathsf{D}(z, \boldsymbol{x})\}$ for a fresh predicate $\mathsf{D}$
13             end
14         end
15         $\mathsf{Rules} := \mathsf{Rules} \cup \{\varphi \to \exists \boldsymbol{v}.\psi\}$
16     end
17 end
18 $\mathsf{Result}_{\mathbb{P}} :=$ Rules with each body object replaced by its predicate as defined in Done
19 $\mathsf{Result}_{\mathbb{D}} :=$ set of all facts $\mathsf{D}(c, d_1, \ldots, d_n)$ for which $\mathbb{D}, \mathsf{Done} \models \mathsf{D}(c, d_1, \ldots, d_n)$
20 return $\langle \mathsf{Result}_{\mathbb{P}}, \mathsf{Result}_{\mathbb{D}} \rangle$
```

Algorithm 1 shows the main part of our procedure, which makes use of some additional notation explained shortly. The algorithm recursively uses rewriting to eliminate defining rules for all (body) objects that are to be denormalised. Todo and Done are sets of defining rules that still need to be rewritten and that already have been rewritten, respectively. Rules is a set of rules obtained from the rewriting. The defining rules needed for the body objects that occur in Rules are always found in Todo $\cup$ Done.

Initially, Rules are the input rules and Todo are the defining rules for their body objects. For each rule in Todo (Line 4), we consider each rewriting using Rules (Line 7) for being added to Rules (Line 15). First, however, we ensure that every body object of newly rewritten rules is defined (Line 8): either we already defined an equivalent object before (Line 9) that we can reuse, or we add a new object definition to Todo (Line 12).

By $\xi[z, \boldsymbol{x}] \equiv \xi'[z', \boldsymbol{x}']$ in Line 9, we express that the two conjunctions are equivalent conjunctive queries, i.e., there is a bijection $\{z\} \cup \boldsymbol{x} \to \{z'\} \cup \boldsymbol{x}'$ that extends to a homomorphism from ξ to ξ', and whose inverse extends to a homomorphism from ξ' to ξ [1]. Checking this could be NP-hard in general, but is possible in subpolynomial time for our special (star-shaped) object conjunctions. By $\xi'[z, \boldsymbol{x}]$ in Line 10, we mean ξ' with $\{z'\} \cup \boldsymbol{x}'$ replaced by $\{z\} \cup \boldsymbol{x}$ according to the bijection that shows equivalence.

If the algorithm terminates, we return the rewritten rules Rules with all body objects replaced using the newly defined D-atoms, and the set of all denormalised facts that follow from the input database. Note that the heads of rules in Rules may already contain denormalisation atoms $\mathsf{D}(z, \boldsymbol{x})$, while the bodies remain normalised during the rewriting.

In Line 19, we do not need to consider rules in Done that contain IDB predicates in their body, so this database denormalisation is simply conjunctive query answering.

Example 4. Applying Algorithm 1 to Example 1, Todo initially contains three defining rules: rule (9), rule $\mathsf{phd}_1(z,x) \wedge \mathsf{phd}_2(z,y_1) \wedge \mathsf{phd}_3(z,y_2) \to \mathsf{D}_{\mathsf{phd}}(z,x,y_1,y_2)$, and rule $\mathsf{sup}_1(z_2,x_1) \wedge \mathsf{sup}_2(z_2,x_2) \to \mathsf{D}_{\mathsf{sup}}(z_2,x_1,x_2)$. The latter two rules contain only EDB predicates in their bodies and therefore have no rewritings: they are moved to Done without adding rules to Rules or Todo. Rule (9) has two rewritings (10) and (11), with the same body objects as the original rule set: all of them are equivalent to objects in Done and can be reused. The algorithm terminates to return four rules: (12) and (13), and analogous denormalisations of the original rules (4) and (5).

Theorem 4. *Consider a database $\mathbb{D}$ and a rule set $\mathbb{P}$, such that Algorithm 1 terminates and returns $\langle \mathsf{Result}_{\mathbb{P}}, \mathsf{Result}_{\mathbb{D}} \rangle$. For any Boolean conjunctive query $\exists \boldsymbol{v}.\varphi[\boldsymbol{v}]$, we have that $\mathbb{D}, \mathbb{P} \models \exists \boldsymbol{v}.\varphi[\boldsymbol{v}]$ iff $\mathsf{Result}_{\mathbb{D}}, \mathsf{Result}_{\mathbb{P}} \models \exists \boldsymbol{v}.\varphi[\boldsymbol{v}]$.*

As usual, this result extends to non-Boolean CQ answering [1]. To prove Theorem 4, one can show the following invariant to hold before and after every execution of the while loop: $\mathbb{D}, \mathbb{P} \models \exists \boldsymbol{v}.\varphi[\boldsymbol{v}]$ iff $\mathbb{D}, \mathsf{Result}_{\mathbb{D}}, \mathsf{Result}_{\mathbb{P}} \models \exists \boldsymbol{v}.\varphi[\boldsymbol{v}]$, where $\mathsf{Result}_{\mathbb{P}}$ and $\mathsf{Result}_{\mathbb{D}}$ are obtained as in Lines 18 and 19 using the current Done. Showing this to hold when the program terminates successfully shows the claim, since $\mathbb{D}$ can be omitted as the rules in $\mathsf{Result}_{\mathbb{P}}$ do not use any EDB predicates from $\mathbb{D}$.

6 Termination of Denormalisation

Although the results of Algorithm 1 are correct, it may happen that the computation does not terminate at all, even in cases where an acceptable rewriting would exist.

Example 5. Consider the rule

$$\mathsf{s}(z_1,x_1) \wedge \mathsf{C}(z_1,x_2) \wedge \mathsf{q}(z_2,x_1) \wedge \mathsf{r}(z_2,x_2) \to \mathsf{C}(z_1,x_1) \tag{14}$$

where s, q, and r are EDB predicates. There are two body objects in (14), where only the first needs rewriting. Rewriting the rule $\mathsf{s}(z_1,x_1) \wedge \mathsf{C}(z_1,x_2) \to \mathsf{D}(z_1,x_1,x_2)$ with (14) leads to a new rule $\mathsf{s}(z_1,x_1) \wedge \mathsf{s}(z_1,x_2) \wedge \mathsf{C}(z_1,x_3) \wedge \mathsf{q}(z_2,x_2) \wedge \mathsf{r}(z_2,x_3) \to \mathsf{D}(z_1,x_1,x_2)$. This rule introduces a new object for object variable z_1. Since the interface now contains three variables $\{x_1,x_2,x_3\}$, it cannot be equivalent to the previous object. A new defining rule is added to Todo, which will subsequently be rewritten to $\mathsf{s}(z_1,x_1) \wedge \mathsf{s}(z_1,x_2) \wedge \mathsf{s}(z_1,x_3) \wedge \mathsf{C}(z_1,x_4) \wedge \mathsf{q}(z_2,x_3) \wedge \mathsf{r}(z_2,x_4) \to \mathsf{D}'(z_1,x_1,x_2,x_3)$. The algorithm therefore does not terminate, and indeed the generated rules are necessary to retain completeness.

As in this example, non-termination of Algorithm 1 is always associated with objects of growing interface. Indeed, for a fixed interface, there are only finitely many non-equivalent objects, so termination is guaranteed. While general (query) rewriting techniques in existential rules tend to have undecidable termination problems, our specific approach allows us to get a more favourable result:

Theorem 5. *It is* P*-complete to decide if Algorithm 1 terminates on a given set of rules. For rule sets that do not contain head atoms of the form $p(x,v)$, where x is a universally quantified variable and v is existentially quantified, the problem becomes* NL*-complete.*

To see why this is the case, let us first observe that non-termination is only caused by rules that use object variables in frontier positions:

Proposition 1. *If object variables do not occur in the frontier of any rule in $\mathbb{P}$, then Algorithm 1 terminates on input $\mathbb{P}$. In particular, this occurs if $\mathbb{P}$ is of the form* $\mathsf{GN}(\mathbb{P}')$.

Indeed, consider a rewriting step as in Definition 7 where we rewrite ρ_1 using ρ_2. If the object variable z in ρ_1 is mapped to an existential variable in ρ_2, i.e., $z\theta \in \boldsymbol{v}$, then no atom of the object in ρ_1 can occur in the body of the rewriting, i.e., φ_1 is empty. Otherwise, there would be an existential (object) variable in the body, which is not allowed by Definition 7. Hence, the body of the rewriting is φ_2, and no new objects are introduced. If all rules are of this form, the overall number of objects that need to be processed is finite and the algorithm must terminate.

Coming back to Theorem 5, we can therefore see that only rewritings using rules with object variables in the frontier need to be considered (we call the associated objects *body frontier object* and *head frontier object*). For investigating termination, we can restrict to "minimal" rewritings that affect only one value y in the rewritten object, i.e., where $\bar{\varphi}_1$ from Definition 7 has the form $p_1(z, y) \wedge \ldots \wedge p_k(z, y)$.

In the (simpler) case that head frontier objects do not have any existentially quantified values, it is even enough to rewrite single attribute-value pairs. A rule with body frontier object $p_1(z, y_1) \wedge \ldots \wedge p_n(z, y_n)$ and head frontier object $q_1(z, x_1) \wedge \ldots \wedge q_m(z, x_m)$ thus gives rise to "replacement rules" of the form $q_i(z, x_i) \mapsto p_j(z, y_j)$ that specify how objects might be rewritten using this rule. This defines a graph on attribute-value pairs of $\mathbb{P}$. Non-termination can be shown to occur exactly if this graph has a cycle along which the interface of the object has increased.

For the latter, we trace the size of the rewritten object's interface during rewriting. Every rewriting with a frontier object may increase or decrease the interface. An increase may occur if the body frontier object contains at least two values in its interface (one interface value preserves size: it is either the frontier value that was unified in the rewriting, or there is no frontier value and the rewritten value was mapped to an existential variable and thereby eliminated). Rule (14), for example, has two interface values, x_1 and x_2, causing non-termination. We can keep track of the interface size in logarithmic space. Cycle detection in the above graph is possible in NL. This shows membership. Hardness is also shown by exploiting the relationship to cycle detection.

Using our understanding of interface-increasing rules as a cause for non-termination, we can also generalise Proposition 1:

Theorem 6. *If every body frontier object that occurs in some rule of $\mathbb{P}$ has an interface of size ≤ 2, then Algorithm 1 terminates on $\mathbb{P}$.*

We have only shown the NL-part of Theorem 5 yet. The general case with existential values is more complicated and we just give the key ideas of the proof in [18]. The problem is that existential values can only be used for rewriting if all attributes of the rewritten object value are found in the head. Hence, it is not enough to trace single attribute-value pairs. P-hardness is shown by reduction from propositional Horn logic entailment, where we encode propositional rules $a \wedge b \rightarrow c$ as $p_a(x, y) \wedge p_b(x, y) \rightarrow p_c(x, y)$ and true propositions a as $t(x, y) \rightarrow p_a(x, y)$. Finally, we add a rule $p_c(x, y) \wedge p_c(x, z) \rightarrow \exists v.t(x, v)$, where c is a proposition. One can show that Algorithm 1 terminates on the resulting rule set if and only if c is *not* entailed from the Horn rules. Membership can use a similar cycle-detection approach, but the construction of the underlying graph now runs in P.

Even Theorem 6 does not guarantee termination for KG linear rules, and indeed our approach may not terminate in this case. To fix this, we need to observe that we can simplify rewriting if all rules contain only one object in their body: using the notation of Definition 7, a *linear rewriting* of rule ρ_1 using ρ_2 is the rule $\varphi_1\theta \wedge \varphi_2 \rightarrow \exists \boldsymbol{v}.\mathsf{D}(x, \boldsymbol{y})\theta$. In words: we are reducing the head to contain only the denormalisation atom, and no other atoms. It is easy to check that the procedure remains complete for KG linear rules.

Theorem 7. *If $\mathbb{P}$ is KG linear, then Algorithm 1, modified to use linear rewriting of rules, terminates and returns a rule set* $\mathsf{Result}_{\mathbb{P}}$ *that is linear.*

It is not hard to see that rewritings of KG linear rules must also be KG linear, showing the second part of the claim. Termination follows since the interface of KG linear rules as obtained during rewriting is bounded by the size of the frontier, which cannot increase when using linear rewriting.

Finally, we remark that our denormalisation shares some similarities with CQ rewriting for existential rules, which is known to be semi-decidable: there is an algorithm that terminates and returns a finite rewriting of a BCQ over a set of rules whenever such a rewriting exists [2]. One may wonder if we could achieve a similar behaviour for Algorithm 1, extending it so that termination is semi-decidable and the algorithm is guaranteed to produce a denormalisation for, e.g., all rule sets that are fus. However, under our assumption that EDB and IDB predicates are separated, the rewritability of BCQs is in fact no longer semi-decidable, not even for plain Datalog. Similar observations have been made for the closely related problem of Datalog *predicate boundedness* [10]. Hence, there is no hope of finding an algorithm that will always compute a denormalisation whenever one exists, even if we cannot decide if this will eventually happen or not. In exchange for this inconvenience, our algorithm also benefits from the separation of IDB and EDB predicates, as it enables us to eliminate defining rules after rewriting them in all possible ways – since IDB predicates cannot occur in the database, this preserves inferences, although it is not semantically equivalent in first-order logic.

7 Frontier Guardedness and Functional Attributes

Our denormalisation procedure can also be applied to KG frontier guarded rules.

Theorem 8. *If $\mathbb{P}$ is KG frontier guarded and Algorithm 1 terminates on $\mathbb{P}$, then the denormalised rule set* $\mathsf{Result}_{\mathbb{P}}$ *is frontier guarded.*

This follows since a KG frontier guarded rule can only have one object variable in its frontier, so that the object in this case must be the guard. Rewriting therefore can only increase the size of the guard, preserving frontier guardedness.

Theorem 8 is still weaker than Theorem 7, since it does not guarantee termination as in the case of KG linear rules. To compensate, we add another mechanism for making termination more likely, following our intuition of viewing conjunctions as "objects".

In typical objects, attributes often can have at most one value. This holds for all objects created when normalising rules. Making this restriction formal could also ensure termination, since the size of each object would be bounded, and the number of possible objects finite. Example 5 shows how a non-terminating case might violate this. The constraint that attributes have at most one value is captured by functional dependencies:

Definition 8. *A* functional dependency *(FD) for attribute p is a rule $p(z, x_1) \wedge p(z, x_2) \rightarrow x_1 \approx x_2$, where $\approx$ is a special predicate that is interpreted as identity relation in all models: $\approx^{\mathcal{I}} = \{\langle \delta, \delta \rangle \mid \delta \in \Delta^{\mathcal{I}}\}$. The functional dependency is an* EDB-FD *if p is an EDB predicate, and an* IDB-FD *otherwise.*

We use built-in equality in this definition, making FDs a special case of *equality generating dependencies* (egds) [1]. Alternatively, $\approx$ could also be axiomatised using Datalog, which turns FDs into regular Datalog rules and $\approx$ into a regular predicate.

Intuitively, we want functional dependencies to apply to some attributes. However, we cannot just introduce FDs as additional rules: query answering is undecidable for the combination of (frontier) guarded existential rules and FDs [15]. Conversely, it is not true that the given rule set *entails* any IDB-FDs, even if some EDB-FDs are guaranteed to hold in the database. Indeed, any model of a set of rules can be extended by interpreting each IDB predicate as a maximal relation (i.e., as an arity-fold cross-product of the domain), resulting in a model that refutes all possible IDB-FDs. Therefore, rather than *asserted* or *entailed* FDs, we are interested in FDs that are *incidental*:

Definition 9. *Consider a set $\mathbb{P}$ of rules and a set $\mathbb{F}$ of EDB-FDs. An IDB-FD for attribute p is* incidental *to $\mathbb{P}$ and $\mathbb{F}$ if, for all databases $\mathbb{D}$ with $\mathbb{D} \models \mathbb{F}$ and for all BCQs φ, we have that $\mathbb{D}, \mathbb{P} \models \varphi$ iff $\mathbb{D}, \mathbb{P} \cup \{p(z, x_1) \wedge p(z, x_2) \rightarrow x_1 \approx x_2\} \models \varphi$. The set of all FDs incidental to $\mathbb{P}$ and $\mathbb{F}$ is denoted* $\mathsf{IDP}(\mathbb{P}, \mathbb{F})$.

In other words, an FD is incidental if we might as well assert it without affecting the answer to any conjunctive query.

Given a set $\mathbb{F}$ of FDs and a conjunction φ of binary atoms of the form $p(x, y)$, we write $\mathbb{F}(\varphi)$ for the conjunction obtained by identifying variables in φ until all FDs in $\mathbb{F}$ are satisfied. This is unique up to renaming of variables. Moreover, let $\theta_{\mathbb{F}(\varphi)}$ denote a corresponding substitution such that $\mathbb{F}(\varphi) = \varphi\theta_{\mathbb{F}(\varphi)}$. For our simple attribute dependencies, this can be computed in polynomial time. Using this notation, we can extend Algorithm 1 to take a given set of FDs into account:

Definition 10. *Let Algorithm $1_{\mathbb{F}}$ be the modification of Algorithm 1 that takes an additional set $\mathbb{F}$ of FDs as an input, and that replaces the rewriting $\varphi \rightarrow \exists \boldsymbol{v}.\psi$ after Line 7 by $\mathbb{F}(\varphi) \rightarrow \exists \boldsymbol{v}.\psi\theta_{\mathbb{F}(\varphi)}$, i.e., which factorises each rewriting using the given FDs before continuing.*

This may help to achieve termination, since the application of FDs may decrease the size of objects to be rewritten next. Our approach shares some ideas with the use of database constraints for optimising query rewriting [23], but the details are different.

Example 6. Consider again the rule of Example 5, and assume that we know that attribute s is functional. Algorithm $1_{\mathbb{F}}$ will again obtain the rewriting $\mathsf{s}(z_1, x_1) \wedge \mathsf{s}(z_1, x_2) \wedge \mathsf{C}(z_1, x_3) \wedge \mathsf{q}(z_2, x_2) \wedge \mathsf{r}(z_2, x_3) \rightarrow D(z_1, x_1, x_2)$. Denoting the body of this rewriting by φ, we find that $\theta_{\mathbb{F}(\varphi)} = \{x_2 \mapsto x_1\}$, so that the rewriting becomes $\mathsf{s}(z_1, x_1) \wedge \mathsf{C}(z_1, x_3) \wedge \mathsf{q}(z_2, x_1) \wedge \mathsf{r}(z_2, x_3) \rightarrow D(z_1, x_1, x_1)$. The object for variable z_1 now is equivalent to the object that has been rewritten in the first step, and so can be replaced by $D(z_1, x_1, x_3)$. The algorithm terminates.

8 Obtaining Incidental FDs

The improved denormalisation of Definition 10 hinges upon the availability of a suitable set of functional dependencies. For EDB predicates, these might be obtained from constraints that have been declared explicitly for the underlying database, or they might even be determined to simply hold in the given data. Example 6 shows that this can already help. In general, however, we would also like to use incidental IDB-FDs. This section therefore asks how they can be computed.

Our first result is negative: it is impossible to determine all incidental FDs even for very restricted subsets of Datalog. This can be shown by reducing from the undecidable problem of deciding non-emptiness of the intersection of two context-free grammars.

Theorem 9. *For a set $\mathbb{P}$ of Datalog rules containing only binary predicates and no constants, a set $\mathbb{F}$ of EDB-FDs, and an IDB-FD σ, it is undecidable if $\sigma \in \mathsf{IDP}(\mathbb{P}, \mathbb{F})$.*

We therefore have to be content with a sound but incomplete algorithm for computing incidental FDs. We use a top-down approach that initially assumes all possible FDs to hold, and then checks which of them might be violated when applying rules, until a fixed point has been reached. This approach is closely related to a work of Sagiv [24, Sect. 9] where the author checks if a given set of existential rules is *preserved non-recursively* by a given Datalog program. We extend this idea from Datalog to existential rules and from non-recursive to (a form of) recursive preservation. For simplicity, we give the algorithm only for checking FD preservation, but it is not hard to extend it to arbitrary rules. We also remark that Theorem 9 settles an open question of Sagiv [24].

Our algorithm tries to discover a violation of an FD by considering a situation where the premise holds (expressed as a CQ $p(z, x_1) \wedge p(z, x_2)$), and then checking all possible ways to derive this situation in one step, using rewriting. If any of the rewritten queries is such that the FD does not follow from the FDs assumed to far, the FD is eliminated.

To check functionality in the presence of existential quantifiers, we first replace existential variables by Skolem terms. The actual check then has to be based on a rewriting of $p(z, x_1) \wedge p(z, x_2)$ where both atoms have been rewritten, which we ensure by renaming the predicates. For the next definition, recall that rewriting conjunctive queries can be achieved like rewriting rules in Definition 7 but dropping the head in all rewritings.

Definition 11. *The* Skolemisation *of rule $\varphi[\boldsymbol{x},\boldsymbol{y}] \rightarrow \exists \boldsymbol{v}.\psi[\boldsymbol{x},\boldsymbol{v}]$ is the rule $\varphi[\boldsymbol{x},\boldsymbol{y}] \rightarrow \psi'[\boldsymbol{x}]$ where ψ' is obtained from ψ by replacing each $v \in \boldsymbol{v}$ by a term $f_v(\boldsymbol{x})$, where f_v is a freshly introduced function symbol. The Skolemisation of all rules in $\mathbb{P}$ is denoted* $\mathsf{skolem}(\mathbb{P})$.

For a conjunction of atoms φ, let $\hat{\varphi}$ be φ with all predicates p replaced by fresh predicates $\hat{p}$. For a rule set $\mathbb{P}$, let $\hat{\mathbb{P}}$ be the set $\{\varphi \rightarrow \exists \boldsymbol{v}.\hat{\psi} \mid \varphi \rightarrow \exists \boldsymbol{v}.\psi \in \mathbb{P}\}$. The one-step rewriting `os-rewrite`$(\varphi, \mathbb{P})$ *is the set of all conjunctions obtained by exhaustively rewriting $\hat{\varphi}$ using rules in* $\mathsf{skolem}(\hat{\mathbb{P}})$*, and where no predicate from $\hat{\varphi}$ occurs.*

Algorithm 2. Algorithm for computing some incidental FDs

```
Input  : rule set $\mathbb{P}$; set $\mathbb{F}$ of EDB-FDs
Output: set $\mathbb{F}_{\mathsf{IDB}}$ of incidental IDB-FDs
1  $\mathbb{F}_{\mathsf{IDB}} := \{p(z, x_1) \wedge p(z, x_2) \to x_1 \approx x_2 \mid p$ an IDB predicate$\}$
2  repeat
3      foreach $p(z, x_1) \wedge p(z, x_2) \to x_1 \approx x_2 \in \mathbb{F}_{\mathsf{IDB}}$ do
4          foreach $\varphi \in$ os-rewrite$(p(z, x_1) \wedge p(z, x_2), \mathbb{P})$ do
5              $y_i$ := the variable that $x_i$ has been mapped to for the rewriting $\varphi$ ($i \in \{1, 2\}$)
6              if $y_1\theta_{(\mathbb{F}\cup\mathbb{F}_{\mathsf{IDB}})(\varphi)} \neq y_2\theta_{(\mathbb{F}\cup\mathbb{F}_{\mathsf{IDB}})(\varphi)}$ then
7                  $\mathbb{F}_{\mathsf{IDB}} := \mathbb{F}_{\mathsf{IDB}} \setminus \{p(z, x_1) \wedge p(z, x_2) \to x_1 \approx x_2\}$
8                  break // continue with next FD in Line 3
9              end
10         end
11     end
12 until $\mathbb{F}_{\mathsf{IDB}}$ has not changed in previous iteration
13 return $\mathbb{F}_{\mathsf{IDB}}$
```

The result of `os-rewrite` is finite, since heads and bodies of $\hat{\mathbb{P}}$ do not share predicates. Our procedure is given in Algorithm 2. It proceeds as explained above checking, given a pair of IDB atoms, every possible derivation for a potential violation of an FD. A violation is detected if two values of an attribute are not necessarily equal based on the current FDs (Line 6). Note that φ may not contain x_1 and/or x_2 since they may be unified during rewriting. We therefore consider the values y_i they have been mapped to (Line 5). As a special case, y_i can be Skolem terms, which typically causes the FD to be violated, unless both x_1 and x_2 are rewritten together and replaced by the same term.

Note that the check in Line 5 uses the set $\mathbb{F}_{\mathsf{IDB}}$, including the FD that is just checked. Intuitively speaking, this is correct since the rewriting approach searches for the first step (in a bottom-up derivation) where an FD would be violated. Initially, when all IDB predicates are empty, all FDs hold.

Theorem 10. *For inputs $\mathbb{P}$ and $\mathbb{F}$, Algorithm 2 returns a set $\mathbb{F}_{\mathsf{IDB}} \subseteq \mathsf{IDP}(\mathbb{P}, \mathbb{F})$ after polynomial time.*

While the algorithm must be incomplete, and in particular cannot detect all FDs for the rules used for our proof of Theorem 9, it can detect many cases of FDs.

Example 7. Consider the following rules, with EDB predicates p and s:

$$\mathsf{p}(x, y) \wedge \mathsf{s}(x, y) \to \mathsf{Q}(x, y) \tag{15}$$

$$\mathsf{s}(x, y) \to \exists v, w.\mathsf{Q}(v, w) \wedge \mathsf{R}(x, v) \wedge \mathsf{R}(x, w) \tag{16}$$

Assume that p is functional. Algorithm 2 first checks the IDB-FD for Q by rewriting $\hat{\mathsf{Q}}(z, x_1) \wedge \hat{\mathsf{Q}}(z, x_2)$. We can rewrite the first atom using rule (15) (mapping z to x and x_1 to y) to obtain $\mathsf{p}(x, y) \wedge \mathsf{s}(x, y) \wedge \hat{\mathsf{Q}}(x, x_2)$. Rewriting $\hat{\mathsf{Q}}(x, x_2)$ using rule (15) with variables renamed to x' and y', we get $\mathsf{p}(x, y) \wedge \mathsf{s}(x, y) \wedge \mathsf{p}(x, y') \wedge \mathsf{s}(x, y')$. Hence $y_1 = y$ and $y_2 = y'$ in Line 5, and these variables are identified since p is functional.

Rewriting $\hat{\mathsf{Q}}(z, x_1) \wedge \hat{\mathsf{Q}}(z, x_2)$ using rule (16) for both atoms, we obtain $\mathsf{s}(x, y) \wedge \mathsf{s}(x, y')$, with original variables replaced by $\{z \mapsto x, x_1 \mapsto f_w(x), x_2 \mapsto f_w(x)\}$ where $f_v(x)$ and $f_w(x)$ are Skolem terms. Again, the FD is preserved. As it is not possible to rewrite one atom with rule (15) and the other with rule (16), we find that Q is functional.

In contrast, functionality for R is violated, since we cannot identify $f_v(x)$ and $f_w(x)$.

9 Discussion and Outlook

Our central observation is that support for ontological modelling and reasoning over knowledge graphs (KGs) is severely lacking. Ontology language features needed for KGs are not supported by mainstream approaches such as OWL and Datalog, and take us outside of known decidable classes of existential rules. Practical tools and methods for modelling and reasoning are even further away. A lot of research is still to be done.

Our work is a first step into this field, focussing on basic language definitions and decidability properties. A core concept of our work is to view some conjunctive patterns as *objects* with *attributes* and *values*, such that existential quantification plays the role of object creation. This leads to a very natural view on existential rules, but it also extends to the data, where objects correspond to groups of triples. We believe that such grouping might also help to improve performance of reasoning with KG-based rules.

Each decidability criterion (acyclicity/fes, bts, rewritability/fus) calls for a different reasoning procedure. For the types of acyclicity we mention, any bottom-up forward chaining inference engine will terminate, even if rules are Skolemised. Rule engines in RDF stores (e.g., Jena) or logic programming tools (e.g., DLV) could be used. Linear rules (and fus in general) are supported by backward-chaining reasoners such as Graal [3]. Interestingly, reasoners for fes and fus do not need to know if and why the rules meet the criteria – it is enough if they do. In particular, rules do not have to be denormalised for reasoning. Denormalisation is only needed to find out which tool to use.

Tools for guarded rules and bts seem to be missing today. They could be implemented by augmenting bottom-up reasoners with additional blocking conditions to ensure termination. Similar ideas are used successfully in OWL reasoning, but generalising them to arbitrary rules will require further research and engineering. Our work may motivate such research by identifying a wider class of rules that would benefit from this.

There are too many connections to other recent works to list, but we highlight some. Ontologies for non-classical data models are currently also studied for key-value stores [21] and for the object database MongoDB [7]. A rule language for declarative programming on KGs was recently proposed in Google's Yedalog [9], and several new rule-based reasoners now support RDF graphs [20,25]. There are numerous works on decidable classes of existential rules. We covered essential approaches, but there remain many others, such as *warded* [14] or *sticky* rules [8], that deserve investigation for KGs.

This diversity of works witnesses a huge current interest in practical data models and rule-based ontologies, but many further works will still be needed for bringing KG-based ontologies to the level of maturity that past semantic technologies have acquired.

Acknowledgements. This work is partly supported by the German Research Foundation (DFG) in CRC 912 (HAEC) and in Emmy Noether grant KR 4381/1-1.

References

1. Abiteboul, S., Hull, R., Vianu, V.: Foundations of Databases. Addison Wesley, Redwood City (1994)
2. Baget, J.F., Leclère, M., Mugnier, M.L., Salvat, E.: On rules with existential variables: walking the decidability line. Artif. Intell. **175**(9–10), 1620–1654 (2011)
3. Baget, J.-F., Leclère, M., Mugnier, M.-L., Rocher, S., Sipieter, C.: Graal: a toolkit for query answering with existential rules. In: Bassiliades, N., Gottlob, G., Sadri, F., Paschke, A., Roman, D. (eds.) RuleML 2015. LNCS, vol. 9202, pp. 328–344. Springer, Heidelberg (2015)
4. Baget, J., Mugnier, M., Rudolph, S., Thomazo, M.: Walking the complexity lines for generalized guarded existential rules. In: Proceedings of the 22nd International Joint Conference on Artificial Intelligence (IJCAI 2011), pp. 712–717 (2011)
5. Belleau, F., Nolin, M., Tourigny, N., Rigault, P., Morissette, J.: Bio2RDF: towards a mashup to build bioinformatics knowledge systems. J. Biomed. Inf. **41**(5), 706–716 (2008)
6. Bollacker, K., Evans, C., Paritosh, P., Sturge, T., Taylor, J.: Freebase: a collaboratively created graph database for structuring human knowledge. In: Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data, pp. 1247–1250. ACM (2008)
7. Botoeva, E., Calvanese, D., Cogrel, B., Rezk, M., Xiao, G.: OBDA beyond relational DBs: a study for MongoDB. In: Proceedings of the 29th International Workshop on Description Logics (DL 2016) (2016)
8. Calì, A., Gottlob, G., Pieris, A.: Towards more expressive ontology languages: the query answering problem. J. Artif. Intell. **193**, 87–128 (2012)
9. Chin, B., von Dincklage, D., Ercegovac, V., Hawkins, P., Miller, M.S., Och, F., Olston, C., Pereira, F.: Yedalog: exploring knowledge at scale. In: 1st Summit on Advances in Programming Languages (SNAPL 2015), pp. 63–78 (2015)
10. Cosmadakis, S.S., Gaifman, H., Kanellakis, P.C., Vardi, M.Y.: Decidable optimization problems for database logic programs (preliminary report). In: Simon, J. (ed.) Proceedings of the 20th Annual ACM Symposium on Theory of Computing (STOC 1988), pp. 477–490. ACM (1988)
11. Cuenca Grau, B., Horrocks, I., Krötzsch, M., Kupke, C., Magka, D., Motik, B., Wang, Z.: Acyclicity notions for existential rules and their application to query answering in ontologies. J. Artif. Intell. Res. **47**, 741–808 (2013)
12. Dantsin, E., Eiter, T., Gottlob, G., Voronkov, A.: Complexity and expressive power of logic programming. ACM Comput. Surv. **33**(3), 374–425 (2001)
13. Das, S., Sundara, S., Cyganiak, R. (eds.): R2RML: RDB to RDF Mapping Language. W3C Recommendation (2012). https://www.w3.org/TR/r2rml/
14. Gottlob, G., Pieris, A.: Beyond SPARQL under OWL 2 QL entailment regime: rules to the rescue. In: Proceedings of the 24th International Joint Conference on Artificial Intelligence (IJCAI 2015), pp. 2999–3007 (2015)
15. Grädel, E.: On the restraining power of guards. J. Symb. Log. **64**(4), 1719–1742 (1999)
16. Hoffart, J., Suchanek, F.M., Berberich, K., Weikum, G.: YAGO2: a spatially and temporally enhanced knowledge base from Wikipedia. J. Artif. Intell. **194**, 28–61 (2013)

17. Krötzsch, M., Rudolph, S.: Extending decidable existential rules by joining acyclicity and guardedness. In: Proceedings of 22nd International Joint Conference on Artificial Intelligence (IJCAI 2011), pp. 963–968 (2011)
18. Krötzsch, M., Thost, V.: Ontologies for knowledge graphs: breaking the rules. Extended technical report, TU Dresden, April 2016. https://iccl.inf.tu-dresden.de/web/TR3029/en
19. Lenzerini, M.: Data integration: a theoretical perspective. In: Popa, L. (ed.) Proceedings of the 21st Symposium on Principles of Database Systems (PODS 2002), pp. 233–246. ACM (2002)
20. Motik, B., Nenov, Y., Piro, R., Horrocks, I., Olteanu, D.: Parallel materialisation of Datalog programs in centralised, main-memory RDF systems. In: Proceedings of the 28th AAAI Conference on Artificial Intelligence, pp. 129–137 (2014)
21. Mugnier, M.L., Rousset, M.C., Ulliana, F.: Ontology-mediated queries for NOSQL databases. In: Proceedings of the 30th AAAI Conference on Artificial Intelligence (2016)
22. W3C OWL Working Group: OWL 2 Web Ontology Language: Document Overview. W3C Recommendation, 27 October 2009. http://www.w3.org/TR/owl2-overview/
23. Rodriguez-Muro, M., Kontchakov, R., Zakharyaschev, M.: Query rewriting and optimisation with database dependencies in ontop. In: Proceedings of the 26th International Workshop on Description Logics (2013)
24. Sagiv, Y.: Optimizing Datalog programs. Technical report CS-TR-86-1132, Stanford University, Department of Computer Science (1986). http://i.stanford.edu/TR/CS-TR-86-1132.html
25. Urbani, J., Jacobs, C., Krötzsch, M.: Column-oriented Datalog materialization for large knowledge graphs. In: Proceedings of the 30th AAAI Conference on Artificial Intelligence, pp. 258–264 (2016)
26. Vrandečić, D., Krötzsch, M.: Wikidata: a free collaborative knowledgebase. Commun. ACM **57**(10), 78–85 (2014)