# Ontologies for Knowledge Graphs: Breaking the Rules*

Markus Krötzsch and Veronika Thost

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

**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 [5] and Wikidata [24]); and in research (e.g., YAGO2 [15] and Bio2RDF [4]).

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 [12]. 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 Figure 1, where $c$ is a fresh element introduced for this tuple, and $\mathsf{s}_1$ to $\mathsf{s}_3$ are binary edge labels used for all tuples of the spouse relation.

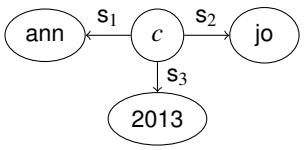


**Fig. 1.** Tuple as Graph

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 [17]. Indeed, ontologies can be used to model semantic relationships between different structures, so that a coherent global view can be obtained.

* This version is an extended technical report with full proofs.

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 spouse(jo, ann, 2013). In a KG, this fact would again be represented by a structure as in Figure 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 [17]. 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 [20], 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 Figure 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], [7], and [10]). 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 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.

We provide extended proof sketches inline. Full formal proofs are in the appendix.

## 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 $\forall \boldsymbol{x}, \boldsymbol{y}.\varphi[\boldsymbol{x}, \boldsymbol{y}] \rightarrow \exists \boldsymbol{v}.\psi[\boldsymbol{x}, \boldsymbol{v}]$ where $\varphi$ and $\psi$ 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 $\varphi$ 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 $\rho = B_1 \wedge \ldots \wedge B_m \rightarrow \exists \boldsymbol{v}.H_1 \wedge \ldots \wedge H_\ell$, let $\mathsf{GN}(\rho)$ be the rule $\bigwedge_{i=1}^{m} \mathsf{GN}(z_i, B_i) \rightarrow \exists \boldsymbol{v}.\exists \boldsymbol{w}.\ \bigwedge_{j=1}^{\ell} \mathsf{GN}(w_j, H_j)$ 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 $\mathsf{P}$ for inferred (IDB) PhD relations:

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

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

The graph normalisation of this rule set is as follows:

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

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

## 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 [10]. 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 $\varphi[\boldsymbol{x}, \boldsymbol{y}] \rightarrow \exists \boldsymbol{v}.\psi[\boldsymbol{x}, \boldsymbol{v}] \in \mathbb{P}$: (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 $\varphi$ and at $\langle q, j\rangle$ in $\psi$; (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 $\varphi$ and there is an existentially quantified variable $v \in \boldsymbol{v}$ at $\langle q, j\rangle$ in $\psi$.*

*$\mathbb{P}$ 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; see Theorem 11 in the appendix.

BCQ entailment for acyclic rule sets is 2ExpTime-complete [10]. Datalog, however, enjoys a lower ExpTime-complete complexity [11], 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 [16, 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 Section 3. For bts, an easy-to-check criterion is (frontier) *guardedness* [2]:

**Definition 3.** *A rule $\varphi[\boldsymbol{x}, \boldsymbol{y}] \rightarrow \exists \boldsymbol{v}.\psi[\boldsymbol{x}, \boldsymbol{v}]$ is* frontier guarded *if $\varphi$ 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 rules 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* [3]; but a procedure for recognising this criterion has not been proposed yet, and it 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 $\varphi$ (or $\psi$) 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 $\varphi$ (or $\psi$), 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 $\varphi$ consists of a single object. It is* pseudo KG frontier guarded *if $\varphi$ contains an object $\xi$ 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 $\mathsf{p}$ 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 $\mathsf{V}$ and $\mathsf{H}$ 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 $\mathsf{D}(z, \boldsymbol{x'})$, while preserving semantics. Note that $\boldsymbol{x'}$ can be limited to the interface of the object with its rule. For example, rule (5) contains the object $\mathsf{P}_1(z_1, x_1) \wedge \mathsf{P}_2(z_1, y_1) \wedge \mathsf{P}_3(z_1, y_2)$ where $y_1$ and $y_2$ do not occur in any other object in body or head. One could therefore replace the object by $\mathsf{D}(z_1, x_1)$, and add a *defining rule*

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

to preserve semantics. We do not need the reverse implication, since $\mathsf{D}$ 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 $\rho_1 : \varphi_1 \wedge \bar{\varphi}_1 \rightarrow \mathsf{D}(x, \boldsymbol{y})$ where $\varphi_1 \wedge \bar{\varphi}_1$ is a single object, and $\rho_2 : \varphi_2 \rightarrow \exists \boldsymbol{v}.(\psi_2 \wedge \bar{\psi}_2) \wedge \xi$ where $\psi_2 \wedge \bar{\psi}_2$ is a single object, so that $\rho_1$ and $\rho_2$ do not share variables. If there is a substitution $\theta$ that maps variables of $\rho_1$ to variables of $\rho_2$ such that $\bar{\varphi}_1\theta = \bar{\psi}_2$, and $\varphi_1\theta$ does not contain any variables from $\boldsymbol{v}$, then the rule $\varphi_1\theta \wedge \varphi_2 \rightarrow \exists \boldsymbol{v}.\mathsf{D}(x, \boldsymbol{y})\theta \wedge \xi$ is a* rewriting of $\rho_1$ using $\rho_2$. *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 $\psi_2$.

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

$$\mathsf{phd}_1(z, x) \wedge \mathsf{phd}_2(z, y_1) \wedge \mathsf{phd}_3(z, y_2) \rightarrow \exists v.\mathsf{D}(v, x) \tag{10}$$

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

Since the $P_i$ are IDB predicates, 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:

$$\mathsf{D}_{\mathsf{phd}}(z, x, y_1, y_2) \rightarrow \exists v.\mathsf{D}(v, x) \tag{12}$$

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

where $\mathsf{D}_{\mathsf{phd}}$ and $\mathsf{D}_{\mathsf{sup}}$ are EDB predicates that need to be defined by denormalising the database, and $\mathsf{D}$ 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. We therefore define $\texttt{rewrite}(\rho_1, \mathbb{P})$ to be the result (least fixed point) of the following recursive process:

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

This approach terminates and $\texttt{rewrite}(\rho_1, \mathbb{P})$ 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 $\varphi \rightarrow \psi_1 \wedge \psi_2$ by two rules $\varphi \rightarrow \psi_1$ and $\varphi \rightarrow \psi_2$ is only correct if $\psi_1$ and $\psi_2$ 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 this 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 ∪ Done.

Initially, Rules are the input rules and Todo are 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 need to 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 $\xi$ to $\xi'$, and whose inverse extends to a homomorphism from $\xi'$ to $\xi$ [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 $\xi'$ 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 rewrit-

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

*Example 4.* Applying Algorithm 1 to Example 1, $\mathsf{Todo}$ initially contains three defining rules: rule (9), rule $\mathsf{phd}_1(z,x) \wedge \mathsf{phd}_2(z,y_1) \wedge \mathsf{phd}_3(z,y_2) \rightarrow \mathsf{D}_{\mathsf{phd}}(z,x,y_1,y_2)$, and rule $\mathsf{sup}_1(z_2,x_1) \wedge \mathsf{sup}_2(z_2,x_2) \rightarrow \mathsf{D}_{\mathsf{sup}}(z_2,x_1,x_2)$. The latter two rules contain only EDB predicates in their bodies and therefore have no rewritings: they are moved to $\mathsf{Done}$ without adding rules to $\mathsf{Rules}$ or $\mathsf{Todo}$. 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 $\mathsf{Done}$ 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 $\mathsf{Done}$. 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

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

where $\mathsf{s}$, $\mathsf{q}$, and $\mathsf{r}$ are EDB predicates. There are two body objects in (14), where only the first needs rewriting. Rewriting the rule $\mathsf{s}(z_1,x_1) \wedge \mathsf{C}(z_1,x_2) \rightarrow \mathsf{D}(z_1,x_1,x_2)$ with (14) leads to a new rule $\mathsf{s}(z_1,x_1) \wedge \mathsf{s}(z_1,x_2) \wedge \mathsf{C}(z_1,x_3) \wedge \mathsf{q}(z_2,x_2) \wedge \mathsf{r}(z_2,x_3) \rightarrow \mathsf{D}(z_1,x_1,x_2)$. 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 $\mathsf{Todo}$, which will subsequently be rewritten to $\mathsf{s}(z_1,x_1) \wedge \mathsf{s}(z_1,x_2) \wedge \mathsf{s}(z_1,x_3) \wedge \mathsf{C}(z_1,x_4) \wedge \mathsf{q}(z_2,x_3) \wedge \mathsf{r}(z_2,x_4) \rightarrow \mathsf{D}'(z_1,x_1,x_2,x_3)$. 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 $\rho_1$ using $\rho_2$. If the object variable $x$ in $\rho_1$ is mapped to an existential variable in $\rho_2$, i.e., $x\theta \in \boldsymbol{v}$, then no atom of the object in $\rho_1$ can occur in the body of the rewriting, i.e., $\varphi_1$ is empty. Otherwise, there would be an existential (object) variable in the body, which cannot be. Hence, the body of the rewriting is $\varphi_2$, 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 $p_1(x, y) \wedge \ldots \wedge p_k(x, y)$.

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 $p_1(x, y_1) \wedge \ldots \wedge p_n(x, y_n)$ and head frontier object $q_1(x, z_1) \wedge \ldots \wedge q_m(x, z_m)$ thus gives rise to replacement rules of the form $q_i(x, z_i) \mapsto p_j(x, y_j)$. 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 $\leq 2$, 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 the appendix. 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 $\rho_1$ using $\rho_2$ 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* [9]. 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 ensure certain termination as given for KG linear rules. To compensate for this deficiency, we add another mechanism for improving termination, which follows our intuition of viewing conjunctions as "objects". In typical objects, many attributes 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: $\approx^{\mathcal{I}} = \{\langle \delta, \delta \rangle \mid \delta \in \Delta^{\mathcal{I}}\}$. We denote such FDs by their attribute $p$. The FD 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 [14]. 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 $\varphi$, 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 set $\mathbb{F}$ of FDs and a conjunction $\varphi$ of binary atoms of the form $p(x, y)$, we write $\mathbb{F}(\varphi)$ for the conjunction obtained by identifying variables in $\varphi$ until all FDs in $\mathbb{F}$ are satisfied. 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 [21], but the details are different.

*Example 6.* Consider again the rule of Example 5, and assume that we know that attribute $\mathsf{s}$ 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 $\varphi$, 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 $\sigma$, 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 [22, Section IX] 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 [22].

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 a 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 $\psi'$ is obtained from $\psi$ by replacing each $v \in \boldsymbol{v}$ by a term $f_v(\boldsymbol{x})$, where $f_v$ is a freshly introduced Skolem function symbol. The Skolemisation of all rules in $\mathbb{P}$ is denoted* $\mathsf{skolem}(\mathbb{P})$.

**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 $\mathbb{F}_{\mathsf{IDB}} \coloneqq \{p(z, x_1) \wedge p(z, x_2) \rightarrow x_1 \approx x_2 \mid p \text{ an IDB predicate}\}$
2 **repeat**
3  **foreach** $p(z, x_1) \wedge p(z, x_2) \rightarrow x_1 \approx x_2 \in \mathbb{F}_{\mathsf{IDB}}$ **do**
4  **foreach** $\varphi \in$ `os-rewrite`$(p(z, x_1) \wedge p(z, x_2), \mathbb{P})$ **do**
5  $y_i \coloneqq$ the variable that $x_i$ has been mapped to for the rewriting $\varphi$ ($i \in \{1, 2\}$)
6  **if** $y_1\theta_{(\mathbb{P} \cup \mathbb{F}_{\mathsf{IDB}})(\varphi)} \neq y_2\theta_{(\mathbb{P} \cup \mathbb{F}_{\mathsf{IDB}})(\varphi)}$ **then**
7  $\mathbb{F}_{\mathsf{IDB}} \coloneqq \mathbb{F}_{\mathsf{IDB}} \setminus \{p(z, x_1) \wedge p(z, x_2) \rightarrow x_1 \approx x_2\}$
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** $\mathbb{F}_{\mathsf{IDB}}$

*For a conjunction of atoms $\varphi$, let $\hat{\varphi}$ be $\varphi$ 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.*

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 $\varphi$ 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 is searching 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:

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

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

where $\mathsf{p}$ and $\mathsf{s}$ are EDB predicates, of which $\mathsf{p}$ is functional. Algorithm 2 first checks the IDB-FD for $\mathsf{Q}$ 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 $\mathsf{p}$ 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 $\mathsf{Q}$ is functional.

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

## 9 Discussion and Outlook

Stepping back from the concrete technical results, our central conceptual contribution is the observation that support for ontological modelling and reasoning over knowledge graphs (KGs) is severely lacking. Ontology language features we need for KGs are not supported by mainstream approaches such as OWL and Datalog, but also take us outside of many known decidable classes of existential rules. Practical tools and methods for modelling and reasoning in this area 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.

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 [19] and for the object database MongoDB [6]. A rule language for declarative programming on KGs was recently proposed in Google's Yedalog [8], and several new rule-based reasoners now support RDF graphs [18,23]. There are numerous works on decidable classes of existential rules. We covered essential approaches, but there remain many others, such as *warded* [13] or *sticky* rules [7], 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.

## References


1. Abiteboul, S., Hull, R., Vianu, V.: Foundations of Databases. Addison Wesley (1994)
2. Baget, J.F., Leclère, M., Mugnier, M.L., Salvat, E.: On rules with existential variables: Walking the decidability line. Artificial Intelligence 175(9–10), 1620–1654 (2011)
3. Baget, J., Mugnier, M., Rudolph, S., Thomazo, M.: Walking the complexity lines for generalized guarded existential rules. In: Proc. 22nd Int. Joint Conf. on Artif. Intell. (IJCAI'11). pp. 712–717 (2011)

4. 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)
5. Bollacker, K., Evans, C., Paritosh, P., Sturge, T., Taylor, J.: Freebase: A collaboratively created graph database for structuring human knowledge. In: Proc. 2008 ACM SIGMOD Int. Conf. on Management of Data. pp. 1247–1250. ACM (2008)
6. Botoeva, E., Calvanese, D., Cogrel, B., Rezk, M., Xiao, G.: OBDA beyond relational DBs: A study for MongoDB. In: Proc. 29th Int. Workshop on Description Logics (DL'16) (2016)
7. Calì, A., Gottlob, G., Pieris, A.: Towards more expressive ontology languages: The query answering problem. J. Artif. Intell. 193, 87–128 (2012)
8. 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'15). pp. 63–78 (2015)
9. Cosmadakis, S.S., Gaifman, H., Kanellakis, P.C., Vardi, M.Y.: Decidable optimization problems for database logic programs (preliminary report). In: Simon, J. (ed.) Proc. 20th Annual ACM Symposium on Theory of Computing (STOC'88). pp. 477–490. ACM (1988)
10. 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. Artificial Intelligence Research 47, 741–808 (2013)
11. Dantsin, E., Eiter, T., Gottlob, G., Voronkov, A.: Complexity and expressive power of logic programming. ACM Computing Surveys 33(3), 374–425 (2001)
12. Das, S., Sundara, S., Cyganiak, R. (eds.): R2RML: RDB to RDF Mapping Language. W3C Recommendation (27 September 2012), available at https://www.w3.org/TR/r2rml/
13. Gottlob, G., Pieris, A.: Beyond SPARQL under OWL 2 QL entailment regime: Rules to the rescue. In: Proc. 24th Int. Joint Conf. on Artif. Intell. (IJCAI'15). pp. 2999–3007 (2015)
14. Grädel, E.: On the restraining power of guards. J. Symb. Log. 64(4), 1719–1742 (1999)
15. 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)
16. Krötzsch, M., Rudolph, S.: Extending decidable existential rules by joining acyclicity and guardedness. In: Proc. 22nd Int. Joint Conf. on Artif. Intell. (IJCAI'11). pp. 963–968 (2011)
17. Lenzerini, M.: Data integration: A theoretical perspective. In: Popa, L. (ed.) Proc. 21st Symposium on Principles of Database Systems (PODS'02). pp. 233–246. ACM (2002)
18. Motik, B., Nenov, Y., Piro, R., Horrocks, I., Olteanu, D.: Parallel materialisation of Datalog programs in centralised, main-memory RDF systems. In: Proc. 28th AAAI Conf. on Artif. Intell. pp. 129–137 (2014)
19. Mugnier, M.L., Rousset, M.C., Ulliana, F.: Ontology-mediated queries for NOSQL databases. In: Proc. 30th AAAI Conf. on Artif. Intell. (2016)
20. OWL Working Group, W.: OWL 2 Web Ontology Language: Document Overview. W3C Recommendation (27 October 2009), available at http://www.w3.org/TR/owl2-overview/
21. Rodriguez-Muro, M., Kontchakov, R., Zakharyaschev, M.: Query rewriting and optimisation with database dependencies in ontop. In: Proc. 26th Int. Workshop on Descr. Logics (2013)
22. Sagiv, Y.: Optimizing Datalog programs. Tech. Rep. CS-TR-86-1132, Stanford University, Department of Computer Science (1986), http://i.stanford.edu/TR/CS-TR-86-1132.html
23. Urbani, J., Jacobs, C., Krötzsch, M.: Column-oriented datalog materialization for large knowledge graphs. In: Proc. 30th AAAI Conf. on Artif. Intell. pp. 258–264 (2016)
24. Vrandečić, D., Krötzsch, M.: Wikidata: A free collaborative knowledgebase. Commun. ACM 57(10) (2014)

# A Appendix of Section 3

We are working with the definitions of acyclicity notions as given by Cuenca Grau et al., which were slightly modified for a more uniform presentation without changing their semantics [10].

In what follows, we further apply the following notions: a variable is called a *frontier* variable if it occurs in both the body and head of a given rule; the Skolemisation of a rule $\rho$ (see Definition 11) is denoted by $\mathsf{skolem}(\rho)$; and *instance* is sometimes used as a synonym for 'database'.

**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.*

We prove the claim separately for each notion of in the following Lemmata 1 (weakly acyclic), 2 (jointly acyclic), 3 (super-weakly acyclic), 5 (model-faithful acyclic), and 6 (acyclic graph of rule dependencies).

**Lemma 1.** *If a rule set $\mathbb{P}$ is weakly acyclic, then so is $\mathsf{GN}(\mathbb{P})$.*

*Proof.* Let $G = (V, E)$ and $G'' = (V'', E'')$ be the dependency graphs of $\mathbb{P}$ and $\mathsf{GN}(\mathbb{P})$, respectively. Define $G' \coloneqq (V', E')$ where $V' = V'' \backslash \{\langle p, 1\rangle \in V''\}$ and $E' = E'' \cap (V' \times V')$, and let $\mu : V \to V'$ be the mapping $\mu : \langle p, i\rangle \mapsto \langle p_i, 2\rangle$.

By Definition 1, $\mu$ is an isomorphism between $G$ and $G'$, which also preserves if an edge is special or not. Since first predicate positions $\langle p, 1\rangle \in V'' \setminus V'$ are never occupied by frontier variables in $\mathsf{GN}(\mathbb{P})$, $G''$ does not contain edges starting in such nodes. Therefore, nodes in $V'' \setminus V'$ cannot be part of cycles in $G''$.

Hence, if $G''$ contains a cycle through a special edge, the cycle must be in $G'$, and, using the isomorphism $\mu$, in $G$. Summing up, if $\mathsf{GN}(\mathbb{P})$ is not weakly acyclic, $\mathbb{P}$ is not weakly acyclic either. □

**Definition 12.** *Let $\mathbb{P}$ be a set of rules, such that no two rules are using the same variable. For a variable $x$, let $\Pi^b_x$ ($\Pi^h_x$) be the set of all positions where $x$ occurs in the body (head) of a – necessarily unique – rule. Now, for any existentially quantified variable, let $\Omega_x$ be the smallest set of positions such that (1) $\Pi^h_x \subseteq \Omega_x$ and (2) $\Pi^h_y \subseteq \Omega_x$, for every universally quantified variable $y$ with $\Pi^b_y \subseteq \Omega_x$.*

*The* existential dependency graph *of $\mathbb{P}$ has the existentially quantified variables of $\mathbb{P}$ as its nodes. There is an edge from $x$ to $y$ if the rule where $y$ occurs contains a frontier variable $z$ with $\Pi^b_z \subseteq \Omega_x$.*

*$\mathbb{P}$ is* jointly acyclic *if its existential dependency graph is acyclic.*

**Lemma 2.** *If a rule set $\mathbb{P}$ is jointly acyclic, then so is $\mathsf{GN}(\mathbb{P})$.*

*Proof.* Consider the existential dependency graph $G$ of the rule set $\mathsf{GN}(\mathbb{P})$. Variables $x$ in the first (object) position of atoms $p_i(x, z)$ in $\mathsf{GN}(\mathbb{P})$ do not occur in frontier positions. Therefore, (1) their sets $\Omega_x$ contain exactly the positions $\langle p_i, 1\rangle$ of $x$ in the unique rule head where it is used; and (2) for every fronter variable $z$, we have $\Pi^b_z \not\subseteq \Omega_x$ and hence $x$ is not the start of any edge in $G$.

For detecting cycles, we can therefore restrict to the graph $G'$ obtained from $G$ by omitting object variables. The rest of the proof proceeds as in Lemma 1, since $G'$ is isomorphic to the existential dependency graph of $\mathbb{P}$. □

Super-weak acyclicity can be defined analogously to joint acyclicity. We overload notation slightly to highlight the similarities.

**Definition 13.** *Let $\mathbb{P}$ be a set of rules where no two rules share any variable, and let $\sigma_{\mathsf{sk}}$ be the substitution used to Skolemise the rules in $\mathbb{P}$. A* place *is a pair $\langle A, i\rangle$ where $A$ is an $n$-ary atom occurring in a rule in $\mathbb{P}$ and $1 \leq i \leq n$. A set of places $P'$* covers *a set of places $P$ if, for each place $\langle A, i\rangle \in P$, a place $\langle A', i'\rangle \in P'$ and substitutions $\sigma$ and $\sigma'$ exist such that $A\sigma = A'\sigma'$ and $i = i'$. Given a variable $x$ occurring in a rule $\rho : \varphi \to \exists v.\psi$, sets of places $\Pi^b_x$, $\Pi^h_x$, and $\Omega^p_x$ are defined as follows:*

- *$\Pi^b_x$ contains each place $\langle p(\boldsymbol{x}), i\rangle$ such that $p(\boldsymbol{x}) \in \varphi$ and $x_i = x$;*
- *$\Pi^h_x$ contains each place $\langle p(\boldsymbol{x})\sigma_{\mathsf{sk}}, i\rangle$ such that $p(\boldsymbol{x}) \in \psi$ and $x_i = x$; and*
- *$\Omega^p_x$ is the smallest set of places such that (1) $\Pi^h_x \subseteq \Omega^p_x$ and (2) $\Pi^h_y \subseteq \Omega^p_x$ for every universally quantified variable $y$ so that $\Omega^p_x$ covers $\Pi^b_y$.*

*The SWA dependency graph $SWA(\mathbb{P})$ of $\mathbb{P}$ contains a vertex for each rule of $\mathbb{P}$, and an edge from a rule $\rho \in \mathbb{P}$ to a rule $\rho' \in \mathbb{P}$ if there is a frontier variable $x'$ of $r'$ and an existentially quantified variable $y$ in the head of $\rho$, such that $\Omega^p_y$ covers $\Pi^b_{x'}$.*

*$\mathbb{P}$ is* super-weakly acyclic (SWA) *if $SWA(\mathbb{P})$ is acyclic.*

**Lemma 3.** *If a rule set $\mathbb{P}$ is super-weakly acyclic, then so is $\mathsf{GN}(\mathbb{P})$.*

*Proof.* We consider the SWA dependency graphs of $\mathbb{P}$ and $\mathsf{GN}(\mathbb{P})$ and show that there is an isomorphism $\mu$ between the two; this directly proves the claim. For rules $\rho$, we define $\mu(\rho) \coloneqq \mathsf{GN}(\rho)$.

As in the proof of Lemma 2, we find that object variables $x$ (those at first positions) in $\mathsf{GN}(\mathbb{P})$ do not occur in frontier positions. Their sets $\Omega^p_x$ are therefore the sets of their original places in $\mathsf{GN}(\mathbb{P})$. They do not give rise to any edges in $SWA(\mathsf{GN}(\mathbb{P}))$, since sets of places at position 1 cannot cover sets of places at position 2, as they are found for all frontier variables.

For value variables $y$ (those at the second predicate position) in $\mathsf{GN}(\mathbb{P})$, we find that $\langle p(\boldsymbol{x}), i\rangle \in \Pi^b_y$ holds w.r.t. $\mathbb{P}$ iff $\langle p_i(z, x_i), 2\rangle \in \Pi^b_y$ holds w.r.t. $\mathsf{GN}(\mathbb{P})$, where $z$ is the variable introduced during normalisation. An analogous correspondence holds for $\Pi^h_y$, and therefore for $\Omega^p_x$. By Definition 13, $\mu$ thus represents an isomorphism. □

The definitions of model-faithful and model-summarising acyclicity are based on some additional notions. The *Skolem chase* is a universal model that is constructed by an exhaustive bottom-up application of all rules of the Skolemisation of a set of existential rules to a given input database. We denote by $\mathbb{D}^0_{\mathbb{P}} = \mathbb{D}, \mathbb{D}^1_{\mathbb{P}}, \mathbb{D}^2_{\mathbb{P}}, \ldots$ the *chase sequence* obtained by applying individual rules of $\mathsf{skolem}(\mathbb{P})$ in a fixed (fair) order, starting with facts of $\mathbb{D}$. The Skolem chase then is $\mathbb{D}^\infty_{\mathbb{P}} = \bigcup_{i\geq 0} \mathbb{D}^i_{\mathbb{P}}$. A more detailed definition of this standard notion is found in the literature [10]. It is easy to see that graph normalisation is largely compatible with the Skolem chase:

**Lemma 4.** *Consider a rule set $\mathbb{P}$ and database $\mathbb{D}$. For every $i \geq 0$, we find that $\mathsf{GN}(\mathbb{D})^i_{\mathsf{GN}(\mathbb{P})}$ is isomorphic to $\mathsf{GN}(\mathbb{D}^i_{\mathbb{P}})$, and therefore that $\mathsf{GN}(\mathbb{D})^\infty_{\mathsf{GN}(\mathbb{P})}$ is isomorphic to $\mathsf{GN}(\mathbb{D}^\infty_{\mathbb{P}})$.*

*Proof.* The required isomorphism is given by mapping the constants $d$ used to translate facts $p(\boldsymbol{c})$ to $p_1(d, c_1), \ldots, p_n(d, c_n)$ in $\mathsf{GN}(\mathbb{D}^i_{\mathbb{P}})$ to the Skolem terms $f(\boldsymbol{c})$ used in groups of facts $p_1(f(\boldsymbol{c}), c_1), \ldots, p_n(f(\boldsymbol{c}), c_n)$ in $\mathsf{GN}(\mathbb{D})^i_{\mathsf{GN}(\mathbb{P})}$. It is easy to see from the definition of the chase and Definition 1 that this yields an isomorphism. □

Furthermore, $I^*_{\mathbb{P}}$ denotes the *critical instance* for a set of rules $\mathbb{P}$, that is, the database that consists of all facts that can be constructed using EDB predicates in $\mathbb{P}$ and a single special fresh constant $*$.[2]

Cuenca Grau et al. define model-faithful acyclicity by means of a program transformation. Our (shorter) definition is based on a characterisation that they establish for their original definition [10, Proposition 5].

**Definition 14.** *A term $t$ is* cyclic *if a function symbol $f$ exists such that some term $f(\boldsymbol{s})$ is a subterm of $t$, and some term $f(\boldsymbol{u})$ is a proper subterm of $f(\boldsymbol{s})$. $\mathbb{P}$ is* model-faithful acyclic *(MFA) w.r.t. a database $\mathbb{D}$ if the Skolem chase $\mathbb{D}^\infty_{\mathbb{P}}$ does not contain a cyclic term. Moreover, $\mathbb{P}$ is* universally MFA *if $\mathbb{P}$ is MFA w.r.t. $I^*_{\mathbb{P}}$.*

**Lemma 5.** *If a rule set $\mathbb{P}$ is universally MFA, then so is $\mathsf{GN}(\mathbb{P})$.*

*Proof.* The claim follows from Lemma 4 if we note that the Skolem terms introduced for existential object variables in rule heads cannot be part of cyclic terms in the Skolem chase, since they only occur on first positions of predicates, where frontier variable do not occur. Therefore, for every database $\mathbb{D}$, we find that $\mathbb{D}^\infty_{\mathbb{P}}$ contains a cyclic term iff $\mathsf{GN}(\mathbb{D})^\infty_{\mathsf{GN}(\mathbb{P})}$ contains a cyclic term. □

The definition of model-summarising acyclicity can be viewed as an approximation of MFA that is obtained by representing Skolem terms with constants. Alternatively, it is a refinement of joint acyclicity, which replaces the coarse estimation of possible positions $\Omega_x$ by a more accurate tracing of the movement of existentially introduced individuals.

**Definition 15.** *For each rule $\rho : \varphi(\boldsymbol{x}, \boldsymbol{y}) \rightarrow \exists v.\psi(\boldsymbol{x}, \boldsymbol{v})$ and each variable $v_i \in \boldsymbol{v}$, let $\mathsf{F}^i_\rho$ be a fresh unary predicate and $c^i_\rho$ be a fresh constant; let $\mathsf{S}$ and $\mathsf{D}$ be fresh binary predicates; and let $\mathsf{C}$ be a fresh nullary predicate. Then, $\mathsf{MSA}(\rho)$ is the following rule, where $\sigma_{\mathsf{MSA}}$ is the substitution that maps each variable $v_i \in \boldsymbol{v}$ to $c^i_\rho$ :*

$$\varphi(\boldsymbol{x}, \boldsymbol{y}) \rightarrow \psi(\boldsymbol{x}, \boldsymbol{v})\sigma_{\mathsf{MSA}} \wedge \bigwedge_{v_i \in \boldsymbol{v}} \left( \mathsf{F}^i_\rho(v_i)\sigma_{\mathsf{MSA}} \wedge \bigwedge_{x_j \in \boldsymbol{x}} \mathsf{S}(x_j, v_i)\sigma_{\mathsf{MSA}} \right).$$

[2] When considering rules with constants, the critical instance should in addition contain all facts built with the constants occurring in the body of a rule in $\mathbb{P}$. This is subsumed by our definition when using auxiliary EDB predicates $O_a$.

*For a rule set $\mathbb{P}$,* $\mathsf{MSA}(\mathbb{P})$ *is the smallest set that contains* $\mathsf{MSA}(\rho)$ *for each rule* $\rho \in \mathbb{P}$*, rules* (17) *and* (18)*, and rule* (19) *instantiated for each predicate* $\mathsf{F}^i_\rho$*:*

$$\mathsf{S}(x,y) \to \mathsf{D}(x,y) \tag{17}$$

$$\mathsf{D}(x,y) \wedge \mathsf{S}(y,z) \to \mathsf{D}(x,z) \tag{18}$$

$$\mathsf{F}^i_\rho(x) \wedge \mathsf{D}(x,y) \wedge \mathsf{F}^i_\rho(y) \to \mathsf{C} \tag{19}$$

$\mathbb{P}$ *is* model-summarising acyclic (MSA) *w.r.t. a database* $\mathbb{D}$ *if* $\mathbb{D} \cup \mathsf{MSA}(\mathbb{P}) \not\models \mathsf{C}$*; furthermore,* $\mathbb{P}$ *is universally MSA if* $\mathbb{P}$ *is MSA w.r.t.* $I^*_\mathbb{P}$.

In essence, the above program transformation together with the rules (17) to (19) ensure that no Skolem constant $c^i_\rho$ is its own successor in a chain of one or more $\mathsf{S}$-facts. Unfortunately, MSA is *not* preserved in graph normalisation.

**Theorem 11.** *There is a universally MSA rule set* $\mathbb{P}$ *such that* $\mathsf{GN}(\mathbb{P})$ *is not universally MSA.*

*Proof.* Our proof uses the separation of EDB and IDB predicates to simplify the construction. Let $\mathbb{P}$ consist of the following rules:

$$\begin{aligned} a(x) &\to \exists w_1, w_2, w_3 . R(w_1, w_2) \wedge R(w_2, w_3) \\ R(x,y) &\to T(x,y) \\ T(x,x) &\to \exists v . P(x,v) \wedge T(v,v) \end{aligned}$$

Then $\mathsf{MSA}(\mathbb{P})$ consists of the rules:

$$\begin{aligned} a(x) &\to R(d_1,d_2) \wedge R(d_2,d_3) \wedge \mathsf{F}_{w_1}(d_1) \wedge \mathsf{F}_{w_2}(d_2) \wedge \mathsf{F}_{w_3}(d_3) \\ R(x,y) &\to T(x,y) \\ T(x,x) &\to P(x,d_4) \wedge T(d_4,d_4) \wedge \mathsf{F}_v(d_4) \wedge \mathsf{S}(x,d_4) \end{aligned}$$

and the rules (17) to (19). The critical instance contains the single fact $a(*)$, and the Skolem chase over this instances produces additional facts $R(d_1,d_2), R(d_2,d_3), \mathsf{F}_{w_1}(d_1)$, $\mathsf{F}_{w_2}(d_2), \mathsf{F}_{w_3}(d_3), T(d_1,d_2), T(d_2,d_3)$. Hence, $\mathsf{MSA}(\mathbb{P})$ is universally MSA.

However, $\mathsf{GN}(\mathbb{P})$ consists of the rules

$$\begin{aligned} a_1(z,x) &\to \exists v_1, v_2, w_1, w_2, w_3 . R_1(v_1,w_1) \wedge R_2(v_1,w_2) \wedge \\ &\qquad\qquad R_1(v_2,w_2) \wedge R_2(v_2,w_1) \\ R_1(z,x) \wedge R_2(z,y) &\to \exists v . T_1(v,x) \wedge T_2(v,y) \\ T_1(z,x) \wedge T_2(z,x) &\to \exists v_1, v_2, v . P_1(v_1,x) \wedge P_2(v_1,v) \wedge T_1(v_2,v) \wedge T_2(v_2,v) \end{aligned}$$

and therefore $\mathsf{MSA}(\mathsf{GN}(\mathbb{P}))$ contains rules of the form

$$\begin{aligned} a_1(z,x) &\to R_1(c_1,d_1) \wedge R_2(c_1,d_2) \wedge R_1(c_2,d_2) \wedge R_2(c_2,d_1) \wedge \mathsf{F}_1(c_1) \wedge \dots \\ R_1(z,x) \wedge R_2(z,y) &\to T_1(c_3,x) \wedge T_2(c_3,y) \wedge \mathsf{F}_3(c_3) \wedge \mathsf{S}(x,c_3) \wedge \mathsf{S}(y,c_3) \\ T_1(z,x) \wedge T_2(z,x) &\to P_1(c_4,x) \wedge P_2(c_4,d_3) \wedge T_1(c_5,d_3) \wedge T_2(c_5,d_3) \wedge \\ &\qquad \mathsf{F}_4(c_4) \wedge \mathsf{F}_5(c_5) \wedge \mathsf{F}'_3(d_3) \wedge \mathsf{S}(x,c_4) \wedge \mathsf{S}(x,c_5) \wedge \mathsf{S}(x,d_3) \end{aligned}$$

and further rules (17) to (19). In this case, the critical instance is $\{a_1(*,*)\}$ and the Skolem chase produces (among others), the facts $R_1(c_1,d_1), R_2(c_1,d_2), R_1(c_2,d_2), R_2(c_2,d_1)$, then $S_1(c_3,d_1), S_2(c_3,d_2), S_1(c_3,d_2), S_2(c_3,d_1)$, which allows us to trigger the third rule for $S_1(c_3,d_2), S_2(c_3,d_2)$. This then leads to facts $T_1(c_5,d_3), T_2(c_5,d_3), \mathsf{F}'_3(d_3)$, allowing us to apply the rule again to obtain the fact $\mathsf{S}(d_3,d_3)$. The rules (17) to (19) then entail $\mathsf{C}$, showing that $\mathsf{MSA}(\mathsf{GN}(\mathbb{P}))$ is not universally MSA. □

Finally, we turn towards another approach towards acyclicity, that is largely complementary to the others discussed above.

**Definition 16.** *The rule dependency relation $\prec \subseteq \mathbb{P} \times \mathbb{P}$ on a set of rules $\mathbb{P}$ is defined as follows. Let $\rho_1 : \varphi_1 \to \exists \boldsymbol{v}_1.\psi_1$ and $\rho_2 : \varphi_2 \to \exists \boldsymbol{v}_2\psi_2$ be rules in $\mathbb{P}$, and let $\mathsf{skolem}(\rho_1) = \varphi_1 \to \psi'_1$ and $\mathsf{skolem}(\rho_2) = \varphi_2 \to \psi'_2$. Then, $\rho_1 \prec \rho_2$ if there exist a set of atoms (possibly using Skolem terms) $\mathbb{D}$; a substitution $\sigma_1$, for all variables in $\mathsf{skolem}(\rho_1)$; and a substitution $\sigma_2$, for all variables in $\mathsf{skolem}(\rho_2)$, such that $\varphi_1\sigma_1 \subseteq \mathbb{D}$, $\varphi_2\sigma_2 \nsubseteq \mathbb{D}$, $\varphi_2\sigma_2 \subseteq \mathbb{D} \cup \psi'_1\sigma_1$, and $\psi'_2\sigma_2 \nsubseteq \mathbb{D} \cup \psi'_1\sigma_1$.*

*$\mathbb{P}$ has an* acyclic graph of rules dependencies *(aGRD) if $\prec$ on $\mathbb{P}$ is acyclic.*

**Lemma 6.** *If a rule set $\mathbb{P}$ has an acyclic graph of rule dependencies, then so has $\mathsf{GN}(\mathbb{P})$.*

*Proof.* The lemma is a direct consequence of the facts that $\mathsf{GN}(\rho_1) \prec \mathsf{GN}(\rho_2)$ implies $\rho_1 \prec \rho_2$ and $\mathsf{GN}(\mathbb{P}) = \bigcup_{\rho\in\mathbb{P}} \mathsf{GN}(\rho)$, by Definition 1.

To see the former, assume $\mathbb{D}$, $\sigma_1$, and $\sigma_2$ are the set of atoms and substitutions showing $\mathsf{GN}(\rho_1) \prec \mathsf{GN}(\rho_2)$. Then, we can construct a set of facts $\mathbb{D}' \coloneqq \{\alpha \mid \text{there is } c \text{ with } \mathsf{GN}(\alpha, c) \in \mathbb{D}\}$ and take $\sigma_1$ and $\sigma_2$ to prove $\rho_1 \prec \rho_2$. For example, assuming $\varphi_1$ and $\varphi'_1$ to be the bodies of rules $\mathsf{GN}(\mathsf{skolem}(\rho))$ and $\mathsf{skolem}(\rho)$, respectively; then, $\varphi_1\sigma_1 \subseteq \mathbb{D}$ clearly implies $\varphi'_1\sigma_1 \subseteq \mathbb{D}'$ given the definition of $\mathbb{D}'$ and Definition 1. The other conditions listed in Definition 16 are satisfied by $\mathbb{D}'$ for the same reasons. □

**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.*

*Proof.* The claim follows from similar reasoning as in the proof of Lemma 1. Since object variables do not occur in frontier positions, first positions of normalised predicates cannot be the starting point of any edge (special or not). The claim follows since all special edges in $\mathsf{GN}(\mathbb{P})$ lead to a first position in a predicate. □

## B Appendix of Section 4

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

*Proof.* If $\mathbb{P}$ is fes or bts, then the core chase is finite or has a bounded treewidth, respectively. The core chase can be obtained by computing a core from the Skolem chase. Core computation commutes with graph normalisation, i.e., the core of $\mathsf{GN}(\mathcal{I})$ is isomoprphic to the graph normalisation of the core of $\mathcal{I}$. The result now follows from Lemma 4, since the core of isomorphic structures is isomorphic, and thus retains key properties such as bounded treewidth and finiteness.

It remains to show the claim for fus. Consider a BCQ $Q$. Let $\varphi$ be a UCQ rewriting of $Q$ over $\mathbb{P}$. We can apply $\mathsf{GN}$ to $\varphi$ in the natural way, by treating it like a disjunction of rule bodies. It is immeiate from the definitions that for any database $\mathbb{D}$, we have $\mathbb{D} \models \varphi$ iff $\mathsf{GN}(\mathbb{D}) \models \mathsf{GN}(\varphi)$. This shows that $\mathsf{GN}(\varphi)$ is a UCQ rewriting of $Q$ over $\mathsf{GN}(\mathbb{P})$. Since $Q$ was arbitrary, this shows that fus is preserved by $\mathsf{GN}$. □

# C Appendix of Section 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 $\mathsf{Result}_{\mathbb{D}}, \mathsf{Result}_{\mathbb{P}} \models \exists \boldsymbol{v}.\varphi[\boldsymbol{v}]$.*

*Proof.* We show that the following invariant holds before and after every execution of the while loop in Algorithm 1: $\mathbb{D}, \mathbb{P} \models \exists \boldsymbol{v}.\varphi[\boldsymbol{v}]$ 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 $\mathsf{Done}$. 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 $\mathbb{D}$.

First, observe that the replacement of body objects by atoms for fresh predicates $\mathsf{D}$ preserves BCQ entailments. Formally, consider a rule $\rho = \varphi \to \exists \boldsymbol{v}.\psi \in \mathbb{P}$ where $\varphi$ contains an object $\xi[z, \boldsymbol{x}]$ with object variable $z$ and interface $\boldsymbol{x}$. Then $\mathbb{P}$ entails the same BCQs as $\mathbb{P}'$, obtained by replacing $\xi[z, \boldsymbol{x}]$ in $\rho$ by $\mathsf{D}(z, \boldsymbol{x})$ and adding a rule $\xi[z, \boldsymbol{x}] \to \mathsf{D}(z, \boldsymbol{x})$. For this, recall that $\mathsf{D}$ is fresh, and in particular does not occur in any of the BCQs considered.

When Algorithm 1 processes rules $\rho = \xi[z, \boldsymbol{x}] \to \mathsf{D}(z, \boldsymbol{x}) \in \mathsf{Todo}$, several things happen:

- $\rho$ is moved from $\mathsf{Todo}$ to $\mathsf{Done}$, which means that the body object $\xi[z, \boldsymbol{x}]$ will henceforth be replaced in $\mathsf{Result}_{\mathbb{P}}$, and facts for $\mathsf{D}$ might be added to $\mathsf{Result}_{\mathbb{D}}$.
- Instead of the defining rule $\rho$, we add the rewritings $\texttt{rewrite}(\rho, \mathsf{Rules})$ to $\mathsf{Rules}$.
- New defining rules for new objects used in $\texttt{rewrite}(\rho, \mathsf{Rules})$ get added to $\mathsf{Todo}$.

If $\xi$ contains only EDB predicates, it is clear that we can replace $\xi[z, \boldsymbol{x}]$ by $\mathsf{D}(z, \boldsymbol{x})$ in rule bodies, and omit $\rho$ if we materialise its (EDB) consequences as done in $\mathsf{Result}_{\mathbb{D}}$.

If $\xi$ also contains IDB predicates, then no facts are added to $\mathsf{Result}_{\mathbb{D}}$ and the defining rule $\rho$ is replaced by its rewritings $\texttt{rewrite}(\rho, \mathsf{Rules})$. We need to show that this does not affect BCQ entailments. It is easy to see that our resolution-based backward chaining is sound. Moreover, it is well known that exhaustive rewriting with so-called piece unifiers is also complete [2]. This can be shown formally by induction over the construction of a chase (canonical model), where we observe that every application of rule $\rho$ relies on at least one premise that was produced by another rule, since the body of $\rho$ contains an IDB atom. Hence, we can find a rewriting that represents the combined application of both rules.

We still need to show that our specific definition of $\texttt{rewrite}(\rho, \mathsf{Rules})$ preserves soundness and completeness, although we do not add all possible piece-rewritings. We consider a rewriting step as specified in Definition 7, using the same notation for rules and conjunctions involved:

– The rewritten rules are $\rho_1 : \varphi_1 \wedge \bar{\varphi}_1 \to \mathsf{D}(x, \boldsymbol{y})$ and $\rho_2 : \varphi_2 \to \exists \boldsymbol{v}.(\psi_2 \wedge \bar{\psi}_2) \wedge \xi$.
– The rewriting is $\varphi_1\theta \wedge \varphi_2 \to \exists \boldsymbol{v}.\mathsf{D}(x, \boldsymbol{y})\theta \wedge \xi$, using substitution $\theta$.

In words, we compute rewritings by replacing the rewriting head object $\psi_2 \wedge \bar{\psi}_2$ entirely in the rewritten rule (i.e., we do not retain the unmatched part $\psi_2$). This is sound, since omitting head atoms leads to a logically weaker statement.

It remains to show that the omission of $\psi_2$ is also complete for our algorithm. The rewritten rule without any omissions would be $\rho' = \varphi_1\theta \wedge \varphi_2 \to \exists \boldsymbol{v}.\psi_2 \wedge \mathsf{D}(x, \boldsymbol{y})\theta \wedge \xi$. We claim that this rule can be replaced by the rules $\rho'_1 = \varphi_1\theta \wedge \varphi_2 \to \exists \boldsymbol{v}.\mathsf{D}(x, \boldsymbol{y})\theta \wedge \xi$ (the rewriting as per Definition 7) and $\rho'_2 = \varphi_1\theta \wedge \varphi_2 \to \exists \boldsymbol{v}.\psi_2 \wedge \xi$ (which is subsumed by $\rho_2$) without loosing entailments. This is clear if $\psi_2$ and $\mathsf{D}(x, \boldsymbol{y})\theta$ do not share existential variables, but it also holds in general, since all rules considered by the algorithm and all rules in the final result $\mathsf{Result}_{\mathbb{P}}$ are such that a body object matches a subset of $\psi_2 \wedge \mathsf{D}(x, \boldsymbol{y})\theta$ if and only if it matches a subset of either $\psi_2$ or it matches $\mathsf{D}(x, \boldsymbol{y})\theta$. This shows that our simplified rewriting preserves completeness of general piece rewriting, and from this the overall claim follows. □

# D Appendix of Section 6

We start by proving Proposition 1, since the insights from this result are also useful for proving Theorem 5.

**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 $\mathsf{GN}(\mathbb{P}')$.*

*Proof.* Given the condition of the while loop and the fact that the result of `rewrite` is finite, we can prove termination by showing that the overall number of rules added to Todo is finite. We proceed by induction and assume $\mathsf{Todo}^0$ to denote the set Todo directly after initialisation. Specifically, we show that the following always hold after initialisation:

– object variables do not occur in the frontier of any rule in Rules;
– every body object occurring in Rules or Todo is isomorphic to a body object occurring in $\mathsf{Todo}^0$ and its interface is a (not necessarily proper) subset of the interface of the latter.

This clearly holds for the base case, before the first iteration of the while loop. To see that it also holds thereafter, consider a rewriting step as in Definition 7 where we rewrite $\rho_1$ using $\rho_2$, here coming from Rules. We assume the object variable $z$ in $\rho_1$ to be mapped to an existential variable in $\rho_2$, i.e., $z\theta \in \boldsymbol{v}$. Consequently, no atom of the object in $\rho_1$ can occur in the body of the rewriting, by the definition of `rewrite`, i.e., $\varphi_1$ is empty; otherwise, there would be an existential (object) variable in the body, which cannot be. Hence, the body of the rewriting is $\varphi_2$ and the frontier does not extend that of $\rho_2$. Especially, new objects are only introduced if the frontier and hence some interfaces have shrunk and corresponding defining rules have not yet been considered. Given the induction hypotheses, we then get that both of the conditions are satisfied. The number of rules added to Todo – in this way – is hence clearly finite. □

**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.*

The proof is provided by the following Lemma 7 to 10. We start with the hardness results, since their proofs also yield some first insights into the chief reasons for non-termination.

**Lemma 7.** *It is* P*-hard to decide if Algorithm 1 terminates on a given set of rules.*

*Proof.* The proof is by reduction from propositional Horn logic entailment, known to be P-complete [11]. More precisely, for a given set $\mathbb{P}$ of propositional rules and proposition $q$, we construct a rule set $\mathbb{P}_q$ and show that Algorithm 1 terminates on one of these sets if and only if $q$ is *not* entailed from $\mathbb{P}$. Without loss of generality, we assume that rules of $\mathbb{P}$ are either of the form $\rightarrow c$ (facts, stating that $c$ is true) or $a \wedge b \rightarrow c$.

Now $\mathbb{P}_q$ is defined to contain the following rules:

$$p_a(x, y) \wedge p_b(x, y) \rightarrow p_c(x, y) \qquad \text{for each } a \wedge b \rightarrow c \in \mathbb{P} \tag{20}$$

$$t(x, y) \rightarrow p_c(x, y) \qquad \text{for each } \rightarrow c \in \mathbb{P} \tag{21}$$

$$p_q(x, y) \wedge p_q(x, z) \rightarrow \exists v.t(x, v) \tag{22}$$

When processing these rules, Algorithm 1 introduces denormalisation predicates for all rule bodies, and then rewrites the resulting defining rules by resolution. Note that all rules contain only a single body object that is also a frontier object, so there will always be only one body object in all resulting rules.

Continued rewriting with a rule of the form (20) and (21) replaces atoms $p_c(x, y)$ by a conjunction of atoms of the form $\bigwedge_{a \in A} p_a(x, y)$ and possibly $t(x, y)$ (occurring at most once), such that $\mathbb{P} \models \bigwedge_{a \in A} a \rightarrow c$. In particular, all of these atoms use the same variables $x$ and $y$.

Rewriting with rule (22) is only possible if $y$ occurs exclusively in the single atom $t(x, y)$. Such rewritings are only obtained by continued rewriting of (conjunctions of) atoms that are entailed by $\mathbb{P}$. In particular, $p_q(x, y)$ can be rewritten to $t(x, y)$ if and only if $\mathbb{P} \models c$.

It remains to show that the algorithm terminates if and only if $\mathbb{P} \models c$ does not hold. Rewriting with rules (20) and (21) does not increase the number of variables, hence can produce only a finite number of non-equivalent body objects. Non-termination therefore occurs only if rule (22) is used for rewriting an arbitrary number of times. Indeed, if this happens, the number of variables in the resulting body object grows, leading to new objects that are not subsumed by any objects constructed before. Since rewriting with rule (22) replaces an atom $t(x, y)$ by atoms $p_q(x, y) \wedge p_q(x, z)$, it can only applied an arbitrary number of times if $p_q(x, y)$ (or, equivalently, $p_q(x, z)$) can be rewritten to $t(x, y)$ (resp. $t(x, z)$), which occurs exactly if $\mathbb{P} \models q$. □

**Lemma 8.** *It is* NL*-hard to decide if Algorithm 1 terminates on a given set of rules that does not contain head atoms of the form $p(x, v)$, where $x$ is a universally quantified variable and $v$ is existentially quantified.*

*Proof.* The proof is a variation of the argument for Lemma 7. We reduce from the known NL-hard problem of source-target reachability in directed graphs. Let $G = \langle V, E \rangle$ be a directed graph with vertices $s, t \in V$.

We use binary predicates $p_a$ for each $a \in V$, and define a rule set $\mathbb{P}_G$ to contain the following rules:

$$p_b(x, y) \to p_a(x, y) \qquad \text{for each } \langle a, b \rangle \in E \tag{23}$$

$$p_s(x, y) \wedge p_s(x, z) \to p_t(x, y) \tag{24}$$

Using an argument similar to the one in the proof of Lemma 7, it is easy to see that Algorithm 1 fails to terminate on $\mathbb{P}_G$ if and only if there is a path from $s$ to $t$ in $G$. □

The NL-hardness proof of Lemma 8 requires only rewriting steps where a single atom $p(x, y)$ is resolved with another single atom in the head. Indeed, rewriting steps where the object $x$ is mapped to an existentially quantified variable cannot contribute to non-termination, as shown in Proposition 1; and rewritings steps where the value $y$ is mapped to an existentially quantified variable cannot occur under the conditions of Lemma 8. We can therefore focus on such simple rewritings of single atoms also for showing membership in NL.

**Lemma 9.** *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, it can be decided in* NL *if Algorithm 1 terminates.*

*Proof.* We construct a directed rewriting graph as follows. The vertices of the graph are binary predicates $p$ that occur in the given set of rules. There is an edge from $p$ to $q$ in the rewriting graph if there are variables $x$ and $y$, such that the atom $p(x, y)$ occurs in the head of a rule where $q(x, y)$ occurs in the body (in this case, $x$ and $y$ must be universally quantified). The edge $p \mapsto q$ is *dangerous* if the interface of the body object $x$ contains three or more variables.

Algorithm 1 fails to terminate if and only if the rewriting graph contains a cycle that includes a dangerous edge. The "if" direction follows since in this case, one can apply the rewritings along the cycle an arbitrary number of times, each time increasing the number of variables in the rewritten object. Since the defining queries of objects cannot be subsumed by queries with fewer distinguished variables, this shows that the algorithm cannot terminate.

The "only if" direction follows from the previous discussion that showed that one can generally restrict to rewriting steps of individual atoms, together with the observation that non-termination can only occur if the objects grow indefinitely (as there is only a finite number of non-equivalent queries over a bounded number of variables).

The claim follows since the rewriting graph is of polynomial size, the presence of individual edges in the rewriting graph can be checked in logarithmic space, and cycles in such a directed graph of polynomial size can be found in NL. □

For the general case, we can also work with a kind of rewriting graph, but we need to consider more complex transitions that are generalisations of the P computations used in the hardness proof of Lemma 7.

**Lemma 10.** *It can be decided in* P *if Algorithm 1 terminates on a given set of rules.*

*Proof.* As before, existentially quantified objects cannot contribute to non-termination, since they replace the rewritten object by an object that was found in the original input, and that is therefore not able to grow indefinitely. However, in contrast to Lemma 9, we now have to consider rewritings of sets of atoms $p_1(x, y) \wedge \ldots \wedge p_n(x, y)$ that use the same object $x$ and value $y$. Sets of such atoms are relevant to decide if a rule with existentially quantified value variable can be used for rewriting. We call such rewritings *existential value rewritings*.

Existential value rewritings always must eliminate all occurrences of the rewritten value $y$. The rewritten (frontier) object $x$ may retain its other values, and it obtains new values from the body. To model this, we can consider objects as collections of values, each characterised by its set of attributes, as follows:

$$\bigwedge_{1 \leq i \leq m_1} p_{1,i}(x, y_1) \wedge \ldots \wedge \bigwedge_{1 \leq i \leq m_n} p_{n,i}(x, y_n)$$

and a rule with frontier object $x$ and an existentially quantified head value $v$ with atoms $\bigwedge_{1 \leq i \leq \ell} p_i(x, v)$ can then be viewed as replacing a value with (a subset of the) attributes $p_i$ by any of the values $\bigwedge_{1 \leq i \leq m_k} p_{k,i}(x, y_k)$. We can restrict to the rewriting of individual values, since rewritings can be restricted to modify single values. As illustrated by the hardness proof in Lemma 7, some computation may be required to see if a given value (characterised by its set of attributes) can be rewritten to a form that allows for an existential value rewriting to be applied.

We formalise these considerations to obtain the required decision procedure. We define an *existential rewriting graph* as follows. The vertices of the graph are sets of predicates $\{p_1, \ldots, p_n\}$ such that $p_1(x, y) \wedge \ldots \wedge p_n(x, y)$ is a maximal conjunction of attributes that occurs in a rule body. The number of such conjunctions is linear in the input. There is a directed edge from $\{p_1, \ldots, p_n\}$ to $\{q_1, \ldots, q_m\}$ if there is a rule $\rho$ such that

1. the body of $\rho$ contains the conjunction $q_1(x, y) \wedge \ldots \wedge q_m(x, y)$,
2. the head of $\rho$ contains an existential variable $v$ and a maximal conjunction $p'_1(x, v) \wedge \ldots \wedge p'_\ell(x, v)$, and
3. it is possible to rewrite $p_1(x, y) \wedge \ldots \wedge p_n(x, y)$ to $p'_1(x, y) \wedge \ldots \wedge p'_\ell(x, y)$ using only rules where $x$ and $y$ are frontier variables.

The edge is *dangerous* if either the existential value rewriting rule $\rho$ or any of the ("universal value") rewriting steps applied for condition (3) is such that the interface of the body object for $x$ contains three or more variables.

Edges of the existential rewriting graph can be computed in polynomial time. Indeed, the check for condition (3) can be performed by starting from $p'_1(x, y) \wedge \ldots \wedge p'_\ell(x, y)$ and saturating it by forward inferences as follows. We interpret $\{p'_1, \ldots, p'_\ell\}$ as a set of logical propositions, and consider each rule with body value $\bigwedge_{1 \leq i \leq m_k} p_{k,i}(x, y_k)$ and head atom $q(x, y_k)$ as a propositional logic rule $p_{k,1} \wedge \ldots \wedge p_{k,m_k} \rightarrow q$. The required saturation is computed as the propositional logic saturation under this set of propositional Horn rules, which can be computed in polynomial time. We consider an inferred

proposition $q$ (that is, each entailed atom $q(x, y)$) dangerous if it was obtained by a dangerous rule application, or (recursively) if it was inferred from at least one dangerous premise. This simple saturation process, too, can be performed in polynomial time.

There is an edge from $\{p_1, \ldots, p_n\}$ to $\{q_1, \ldots, q_m\}$ in the existential rewriting graph if we find a rule $\rho$ as in the definition such that the saturation of $\{p'_1(x, y), \ldots, p'_\ell(x, y)\}$ contains all atoms of $\{p_1(x, y), \ldots, p_n(x, y)\}$. The edge is dangerous if at least one of the atoms $\{p_1(x, y), \ldots, p_n(x, y)\}$ is dangerous in the saturation.

Now Algorithm 1 fail to terminate if and only if one of the following is true: (1) the rewriting graph of Lemma 9 contains a cycle with a dangerous edge; or (2) the existential rewriting graph contains a cycle with a dangerous edge. These conditions can be checked in polynomial time, since, in particular, the existential rewriting graph can be constructed in polynomial time. □

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

*Proof.* This is an immediate consequence of the proof of Lemma 10. If every body frontier object that occurs in some rule of $\mathbb{P}$ has an interface of size $\leq 2$, then the rewriting graph and the existential rewriting graph do not contain any dangerous edges, so termination is clear. □

**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.*

*Proof.* Taking the relevant Definitions 6 and 7 into account, it is not hard to see that rewritings of KG linear rules must also be KG linear, showing the second part of the claim. The former specifically holds because defining rules, which are KG linear, are the only rules rewritten. Hence, by rewriting them using KG linear rules the condition $\bar{\varphi}_1\theta = \bar{\psi}_2$ in Definition 7 ensures that the resulting rule is also KG linear. Thus, all rules processed during rewriting are KG linear.

We show the termination result based on the latter observation. It means that, by Definition 6, the object variable of a body object obtained during rewriting cannot occur as a value. Definition 5 then yields that the interface of such a body object is bounded by the size of the frontier of that rule (i.e., the rule obtained by rewriting). Since this frontier equals the one of the rule rewritten when using *linear* rewriting, the number of non-equivalent objects created is bounded.

## E Appendix of Section 7

**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.*

*Proof.* KG frontier guarded rules require that object variables do not occur in value positions. Therefore any such rule has at most one object variable in its frontier, and the body object of this variable is the guard object. Clearly, rewriting with such rules can lead to an increase of the size of the body guard object, but it cannot introduce body frontier variables that are not guarded. □

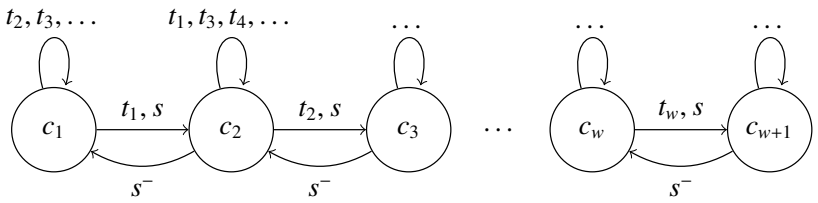


**Fig. 2.** The representation of a word $t_1 t_2 \ldots t_w$ in a database

# F Appendix of Section 8

For the proof of Theorem 9 it should be noted that the omission of constants does make the argument more complicated. It is not possible to refer to the encoding of constants $c$ with unary predicates $O_c$ since we quantify over all databases for the definition of incidental FDs, including those where $O_c$ may not contain exactly one fact.

**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 $\sigma$, it is undecidable if $\sigma \in \mathsf{IDP}(\mathbb{P}, \mathbb{F})$.*

*Proof.* The proof is by reduction of the disjointness problem for context-free languages, which is known to be undecidable. More precisely, we reduce from the following problem:

> Given two context-free grammars $G_1$ and $G_2$ that define languages $\mathcal{L}_1$ and $\mathcal{L}_2$ over an alphabet $\Sigma$, is there a word $w \in \mathcal{L}_1 \cap \mathcal{L}_2$ such that $|w| > 1$?

The undecidability of this problem follows from the undecidability of the disjointness (proof by contradiction: if the problem for words of length greater 1 would be decidable, disjointness would be decidable in general, since the problem is clearly decidable for the finite number of words of length $\leq 1$).

Without loss of generality, we assume that the non-terminal symbols of $G_1$ and $G_2$ are disjoint. Our encoding uses the following predicate symbols: an IDB predicate $P$; EDB predicates $s$ and $s^-$; EDB predicates $t_i$ for each terminal symbol $t_i \in \Sigma$; and IDB predicates $N_i$ for each non-terminal symbol $N_i$ occurring in $G_1$ or $G_2$. We use $S_1$ and $S_2$ to denote the starting non-terminal symbol of $G_1$ and $G_2$, respectively. We require all EDB predicates to be functional, i.e., $\mathbb{F}$ is the set of all functional dependencies for $s$, $s^-$, and $t_i \in \Sigma$.

Words $t_1 t_2 \cdots t_w$ are represented by databases as sketched in Fig. 2. For a letter $t \in \Sigma$ and variables $x$ and $y$, let $\varphi_t[x, y]$ denote the formula $t(x, y) \wedge s(x, y) \wedge s^-(y, x) \wedge \bigwedge_{t' \in \Sigma \setminus \{t\}} t'(x, x)$. Moreover, for a non-terminal $N$, let $\varphi_N[x, y] \coloneqq N(x, y)$. We define $\mathbb{P}$ to contain all rules of the form

$$\varphi_{Z_1}[x_1, x_2] \wedge \ldots \wedge \varphi_{Z_n}[x_n, x_{n+1}] \rightarrow N(x_1, x_{n+1}) \tag{25}$$

for all grammar rules $(N ::= Z_1 \cdots Z_n) \in G_1 \cup G_2$, where $Z_i$ might be terminal or non-terminal symbols. Moreover, $\mathbb{P}$ contains the rules

$$s(x, y) \to P(x, y) \tag{26}$$

$$S_1(x, y) \wedge S_2(x, y) \to P(x, y) \tag{27}$$

where $S_1$ and $S_2$ are the starting symbols of $G_1$ and $G_2$, respectively.

We claim that there is an incidental functional dependency for $P$ in $\mathsf{IDP}(\mathbb{P}, \mathbb{F})$ if and only if there is *no* word $w \in \mathcal{L}_1 \cap \mathcal{L}_2$ such that $|w| > 1$.

($\Rightarrow$) We show the contrapositive. Assume that there is a word $w = t_1 t_2 \cdots t_{|w|} \in \mathcal{L}_1 \cap \mathcal{L}_2$ such that $|w| > 1$. Let $\mathbb{D}_w$ be the database $\{\varphi_{t_i}[c_i, c_{i+1}] \mid 1 \leq i \leq |w|\}$, where $c_j$ are constants (and we slightly extend the notation $\varphi_t[c, d]$ to represent sets of ground facts). Since $w \in \mathcal{L}_1 \cap \mathcal{L}_2$, rules (25) imply $\mathbb{D}_w, \mathbb{P} \models S_1(c_1, c_{|w|+1})$ and $\mathbb{D}_w, \mathbb{P} \models S_2(c_1, c_{|w|+1})$. Hence, by rule (27), we find $\mathbb{D}_w, \mathbb{P} \models P(c_1, c_{|w|+1})$. Rule (27) further implies $\mathbb{D}_w, \mathbb{P} \models P(c_1, c_2)$. Since $|w| > 1$, we have $c_2 \neq c_{|w|+1}$.

Now let $Q$ be the conjunctive query $Q = \exists \boldsymbol{x}.s(x_1, x_2) \wedge s(x_1, x_2) \wedge \ldots \wedge s(x_{|w|}, x_{|w|+1}) \wedge x_2 \approx x_{|w|+1}$. Clearly, $\mathbb{D}_w, \mathbb{P} \not\models Q$, since $\approx$ is interpreted the identity relation over $\mathbb{D}_w, \mathbb{P}$ and $c_2 \neq c_{|w|+1}$. However, $\mathbb{D}_w, \mathbb{P} \cup \{P(z, x_1) \wedge P(z, x_2) \to x_1 \approx x_2\} \models Q$. Therefore, the FD for $P$ is not incidental if a word like $w$ exists.

($\Leftarrow$) We show the contrapositive. Assume that the FD for $P$ is not incidental. This occurs exactly if there is a database $\mathbb{D}$ that satisfies the FDs in $\mathbb{F}$ and constants $c_0, c_1, c_2$ with $c_1 \neq c_2$, such that $\mathbb{D}, \mathbb{P} \models P(c_0, c_1)$ and $\mathbb{D}, \mathbb{P} \models P(c_0, c_2)$. Since $s$ is functional, rule (26) can only entail at most one of the required $P$ facts, and the other (or both) must be entailed by rule (27). Since rules (25) contain a direct translation of the grammar rules, it is clear that facts $S_i(d_1, d_n)$ for $i \in \{1, 2\}$ are only entailed if the database contains a structure $\varphi_{t_1}(d_1, d_2), \ldots, \varphi_{t_n}(d_n, d_{n+1})$ with $t_1 \cdots t_n \in \mathcal{L}_i$. A priori, it is not clear that this structure is the same for both $S_1(d_1, d_n)$ and $S_2(d_1, d_n)$ (or even that $n$ is the same in both cases) – however, we can conclude this from the following observations:

- Since $\varphi_t(x, y)$ contains $s(x, y)$ and $s^-(y, x)$, and both $s$ and $s^-$ are functional, we find that there is a unique chain from $d_1$ to $d_n$ in $\mathbb{D}$.
- Since $\varphi_t(x, y)$ contains $t'(x, x)$ for all $t' \in \Sigma \setminus \{t\}$, where $t'$ is functional, we find that, for each pair of elements $d_i, d_{i+1}$, there is exactly one fact $t(d_i, d_{i+1}) \in \mathbb{D}$.

This shows that $\mathbb{D}$ must contain a chain $\varphi_{t_1}(d_1, d_2), \ldots, \varphi_{t_n}(d_n, d_{n+1})$ with $t_1 \cdots t_n \in \mathcal{L}_1 \cap \mathcal{L}_2$, which leads to the entailment of the fact $P(d_1, d_{n+1})$. Moreover, for violating the FD for $P$, there must be one such chain with $n > 1$, which shows the claim. □

The next lemma can be used to show that an FD is incidental. The result holds for any universal model, e.g., for the Skolem chase.

**Lemma 11.** *Consider a rule set $\mathbb{P}$, a set of EDB-FDs $\mathbb{F}$, and an IDB-FD $\varphi$. If, for all databases $\mathbb{D}$ that satisfy $\mathbb{F}$, there is a universal model $\mathcal{I}$ of $\mathbb{D}$ and $\mathbb{P}$ such that $\mathcal{I} \models \varphi$, then $\varphi \in \mathsf{IDP}(\mathbb{P}, \mathbb{F})$.*

*Proof.* For all BCQs $Q$, we have that $\mathbb{D}, \mathbb{P} \models Q$ implies $\mathbb{D}, \mathbb{P} \cup \{\varphi\} \models Q$. We need to show the converse to show that $\varphi \in \mathsf{IDP}(\mathbb{P}, \mathbb{F})$. If $\mathbb{D}, \mathbb{P} \cup \{\varphi\} \models Q$, then $\mathcal{I} \models Q$, since $\mathcal{I} \models \mathbb{D}$ and $\mathcal{I} \models \mathbb{P} \cup \{\varphi\}$ by definition. Since $\mathcal{I}$ is a universal model of $\mathbb{D}$ and $\mathbb{P}$, all BCQs entailed by $\mathcal{I}$ are also entailed by $\mathbb{D}$ and $\mathbb{P}$, so $\mathbb{D}, \mathbb{P} \models Q$ as required. □

**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.*

*Proof.* We start with the termination result. The size of the initial set $\mathbb{F}_{\mathsf{IDB}}$ is linear in the number of IDB predicates. Since FDs are only removed from $\mathbb{F}_{\mathsf{IDB}}$ (i.e., the set is never extended), there are at most linearly repetitions of the repeat-until loop of line 12. This already shows termination. For each repetition, the loop in 3 is executed a linear number of times. The inner loop in line 4 is executed at most once for each of the quadratically many possible rewritings in `os-rewrite`. The entailments of the assumed FDs that are needed for the check in line 6 can also be computed in polynomial time, leading to a polynomial time algorithm overall.

For soundness, we show that each FD in the returned set $\mathbb{F}_{\mathsf{IDB}}$ holds in the Skolem chase of $\mathbb{P}$ and any database $\mathbb{D}$ that satisfies $\mathbb{F}$. The claim then follows from Lemma 11.

For a contradiction, assume that there is an FD $\psi \in \mathbb{F}_{\mathsf{IDB}}$ such that $\psi \notin \mathsf{IDP}(\mathbb{P}, \mathbb{F})$. By Lemma 11, there is a database $\mathbb{D}$ that satisfies $\mathbb{F}$, such that $\psi$ is not satisfied in the Skolem chase for $\mathbb{D}$ and $\mathbb{P}$. Let $\mathcal{I}_0, \mathcal{I}_1, \ldots$ denote the sequence of interpretations obtained when computing the Skolem chase, where each $\mathcal{I}_{i+1}$ is obtained from $\mathcal{I}_i$ by applying a single (Skolemised) rule. Initially, $\mathcal{I}_0 = \mathbb{D}$, and thus all IDB-FDs are satisfied by $\mathcal{I}_0$. Let $k$ be the smallest index such that $\mathcal{I}_k \not\models \psi$. Without loss of generality we can assume that $\psi$ was chosen in such a way that $k$ is the smallest index of all $\psi' \in \mathbb{F}_{\mathsf{IDB}} \setminus \mathsf{IDP}(\mathbb{P}, \mathbb{F})$, i.e., it is the first among the wrongly computed incidental FDs that is refuted in the Skolem chase.

Let $S$ be the predicate that is asserted to be functional by $\psi$. Then $\mathcal{I}_k$ contains two facts $S(c, t_1)$ and $S(c, t_2)$ with $t_1 \neq t_2$. These facts have been inferred by two (not necessarily distinct) rules, $\rho_1$ and $\rho_2$, from the facts present in some $\mathcal{I}_i$ with $i < k$. There is a one-step rewriting $\varphi \in \texttt{os-rewrite}(S(z, x_1) \wedge S(z, x_2), \mathbb{P})$ where $S(z, x_1)$ is rewritten using $\rho_1$ and $S(z, x_2)$ is rewritten using $\rho_2$.

Let $\mathbb{F}_{\mathsf{IDB}}$ be the output of the algorithm. Then $\mathbb{F}_{\mathsf{IDB}}$ has been considered in the last repetition of the repeat-until loop starting in line 12. We claim that, for the rewriting $\varphi$ as above, the check in line 6 succeeded, i.e., the FD $\psi$ must have been eliminated at this stage, which yields the required contradiction. Indeed, since $k$ was chosen to be smallest, $\psi$ was the first wrong FD to be disproved in the Skolem chase, and the rules $\rho_1$ and $\rho_2$ have produced $S(c, t_1)$ and $S(c, t_2)$ from a database where all IDB predicates in $\mathbb{F}_{\mathsf{IDB}}$ (including $S$) were still functional. The fact that the rules have still produced the facts shows that the FDs cannot entail $t_1$ to be equal to $t_2$. Since $\varphi$ is the most general rewriting using these rules, we must have $y_1\theta_{(\mathbb{P} \cup \mathbb{F}_{\mathsf{IDB}})(\varphi)} \neq y_2\theta_{(\mathbb{P} \cup \mathbb{F}_{\mathsf{IDB}})(\varphi)}$ in line 6. □