# Sound, Complete and Minimal UCQ-Rewriting for Existential Rules

Mélanie König, Michel Leclère, Marie-Laure Mugnier, Michaël Thomazo

# Sound, Complete and Minimal UCQ-Rewriting for Existential Rules

Mélanie König[1], Michel Leclère[1], Marie-Laure Mugnier [1] and Michaël Thomazo*[2]

[1]University Montpellier 2, France
[2]TU Dresden, Germany

**Abstract**

We address the issue of Ontology-Based Data Access, with ontologies represented in the framework of existential rules, also known as Datalog±. A well-known approach involves rewriting the query using ontological knowledge. We focus here on the basic rewriting technique which consists of rewriting the initial query into a union of conjunctive queries. First, we study a generic breadth-first rewriting algorithm, which takes any rewriting operator as a parameter, and define properties of rewriting operators that ensure the correctness of the algorithm. Then, we focus on piece-unifiers, which provide a rewriting operator with the desired properties. Finally, we propose an implementation of this framework and report some experiments.

## 1 Introduction

We address the issue of Ontology-Based Data Access, which aims at exploiting knowledge expressed in ontologies while querying data. In this paper, ontologies are represented in the framework of existential rules [BLMS11, KR11], also known as Datalog± [CGK08, CGL09]. Existential rules allow one to assert the existence of new unknown individuals, which is a key feature in an open-world perspective, as for instance in incomplete databases [CLR03]. These rules are of the form *body* $\rightarrow$ *head*, where the body and the head are conjunctions of atoms (without functions) and variables that occur only in the head are *existentially* quantified. They generalize lightweight description logics (DLs), which form the core of the tractable profiles of OWL2 [OWL09].

The general query answering problem can be expressed as follows: given a knowledge base (KB) $\mathcal{K}$ composed of a set of facts —or data— and an ontology (a set of existential rules here), and a query $Q$, compute the set of answers to $Q$ in $\mathcal{K}$. In this paper, we consider Boolean conjunctive queries (Boolean CQs or BCQs). Note however that all our results are easily extended to non-Boolean conjunctive queries as well as to unions of conjunctive queries. The fundamental problem, called BCQ entailment hereafter, can be recast as follows: given a KB $\mathcal{K}$, composed of facts and existential rules, and a Boolean conjunctive query $Q$, is $Q$ entailed by $\mathcal{K}$?

*This work was done when M. Thomazo was a PhD student at University Montpellier 2.

BCQ entailment is undecidable for general existential rules (e.g., [BV81, CLM81], on the implication problem for tuple-generating dependencies, which have the same form as existential rules). There has been an intense research effort aimed at finding decidable subsets of rules that provide good tradeoffs between expressivity and complexity of query answering (see e.g., [Mug11] for a survey on decidable classes of rules). These decidable rule fragments overcome some of the limitations of DLs. In particular, they have unrestricted predicate arity, while DLs consider unary and binary predicates only, which allows one for a natural coupling with database schemas, in which relations may have any arity; moreover, adding information, such as data provenance, is made easier by the unrestricted predicate arity, since this information can be added as a new predicate argument.

There are two main approaches to solve BCQ entailment, which are linked to the classical paradigms for processing rules, namely forward and backward chaining, as illustrated by the next example.

**Example 1** *Let us consider data on movies, with unary relations* movie *and* actor*, and a binary relation* play *(intuitively,* play$(x, y)$ *means that "$x$ plays a role in $y$"). Let $Q$ be a query asking if a given person, whose identifier is $B$, plays a role somewhere, i.e., $Q = \exists y\ play(B, y)$. Let $R$ be an existential rule expressing that "every actor plays a role in some movie", i.e., $\forall x(actor(x) \rightarrow \exists y(play(x, y) \wedge movie(y)))$. Assume that the data contain* actor($B$)*. If $Q$ is asked on these data, the answer is no. However, the rule allows to infer that actor $B$ plays in a movie, thus the answer to $Q$ should be yes. Rule $R$ can be used in a forward manner, i.e., it can be applied to the data: then, the knowledge $\exists y_0(play(B, y_0) \wedge movie(y_0))$ is added, where $y_0$ is a new variable. Query $Q$ can be mapped to the enriched data, which allows to answer positively. Now, $R$ can also be used in a backward manner, i.e., to rewrite $Q$, which yields the new query $Q'$ =* actor*($B$). This query can be mapped to the (initial) data, which provides the positive answer.*

Both approaches can be seen as ways of reducing the problem to a classical database query answering problem by eliminating the rules, see Figure 1. The first approach consists in applying the rules to the data, thus materializing entailed facts into the data. Then, $Q$ is entailed by $\mathcal{K}$ if and only if it can be mapped to this materialized database. This approach is applicable either when the forward chaining procedure stops "naturally" (see [GHK$^+$13] for a survey on these cases), or when it stops by taking some parameters into consideration, typically the size of the query [CGK08, LMTV12]. The second approach consists in using the rules to rewrite the query into a first-order query (typically a union of conjunctive queries [CGL$^+$07, PUHM09, GOP11, VSS14, RMC12]) or a non-recursive Datalog program [RA10, OP11, GS12]. Then, $Q$ is entailed by $\mathcal{K}$ if and only if the rewritten query is entailed by the initial database.

Materialization has the advantage of enabling efficient query answering but may be not appropriate for data size, data access rights or data maintenance reasons. Query rewriting has the advantage of avoiding changes in the data, however its drawback is that the rewritten query may be large, even exponential in the size of initial query, hence less efficiently processed, at least with current database techniques. Finally, techniques combining both approaches have been developed, in particular so-called combined approach [LTW09, KLT$^+$11] for lightweight description logics, as well as a similar algorithm for a large class of existential rules [TBMR12].

In this paper, we focus on rewriting techniques, and more specifically on rewriting the initial conjunctive query $Q$ into a union of conjunctive queries, that we will see as

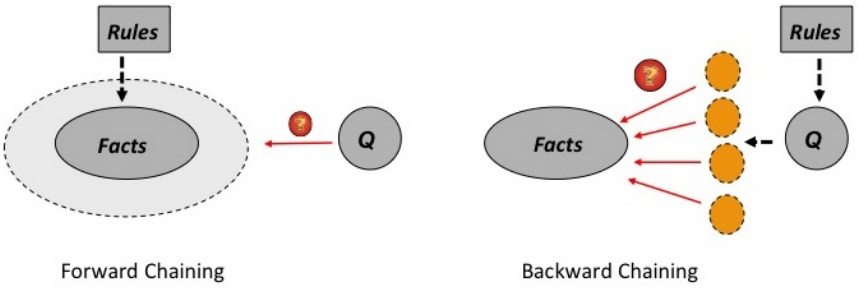


Figure 1: Forward / Backward Chaining

a *set* of conjunctive queries. This set is called a *rewriting set* of $Q$ and each element of a rewriting set is called a *rewriting*. While most previously cited work focuses on specific rule sublanguages (mostly DL-Lite, linear and sticky existential rules), we consider general existential rules. This means that our algorithm does not make any syntactic assumption on the input set of rules, but will terminate only in some cases (so-called *finite unification sets* of rules, see hereafter).

The goal is to compute a rewriting set both *sound* (if one of its elements maps to the initial database, then $\mathcal{K}$ entails $Q$) and *complete* (if $\mathcal{K}$ entails $Q$ then there is an element that maps to the initial database). Minimality may also be a desirable property. In particular, let us consider the generalization relation (a preorder) induced on Boolean conjunctive queries by homomorphism: we say that $Q_1$ is more general than $Q_2$ if there is a homomorphism from $Q_1$ to $Q_2$; it is well-known that the existence of such a homomorphism is equivalent to the following property: for any set of facts $F$, if the answer to $Q_2$ in $F$ is positive, then so is the answer to $Q_1$. We point out that any sound and complete rewriting set of a query $Q$ remains sound and complete when it is restricted to its most general elements. Since BCQ entailment is undecidable, there is no guarantee that such a *finite* set exists for a given query and general existential rules. A set of existential rules ensuring that a finite sound and complete set of most general rewritings exists for *any* query is called a *finite unification set* (*fus*) [BLMS11]. The *fus* property is not recognizable [BLMS11], but several easily recognizable *fus* classes have been exhibited in the literature: atomic-body rules [BLMS09], also known as linear TGDs [CGL09], multi-linear TGDs [CGL12], sticky(-join) rules [CGP10, GOP11], weakly-recursive rules [CR12] and sets of rules with an acyclic graph of rule dependencies [BLMS09]. By definition, the fus property is a specific case of *first-order rewritability*, which means that the set of rules allows to rewrite any CQ into a (sound and complete) first-order query; it is suspected that both properties are actually equivalent, however, to the best of our knowledge, no proof of this result has been published.

**Paper contributions.** We start from a generic algorithm which, given a BCQ and a set of existential rules, computes a rewriting set. This task can be recast in terms of exploring a potentially infinite space of queries, composed of the initial conjunctive query and its (sound) rewritings, structured by the generalization preorder. The algorithm explores this space in a breadth-first way, with the aim of computing a complete rewriting set. It maintains a rewriting set $\mathcal{Q}$ and iteratively performs the following

tasks: (1) generate all the one-step rewritings from unexplored queries in $\mathcal{Q}$; (2) add these rewritings to $\mathcal{Q}$ and update $\mathcal{Q}$ in order to keep only incomparable most general elements. A *rewriting operator* is a function that, given a query and a set of rules, returns the one-step rewritings of this query. Note that it may be the case that the set of sound rewritings of the query is infinite while the set of its most general sound rewritings is finite. It follows that a simple breadth-first exploration of the rewriting space is not sufficient to ensure finiteness of the process, even for *fus* rules; one also has to maintain a set of the most general rewritings. This algorithm is generic in the sense that it is not restricted to a particular kind of existential rules nor to a specific rewriting operator (without guarantee of termination though).

This algorithmic scheme established, we then asked ourselves the following questions:

1. Assuming that the algorithm outputs a finite sound and complete set rewritings, composed of pairwise incomparable queries, is this set of minimal cardinality, in the sense that no sound and complete rewriting set produced by any other algorithm can be strictly smaller?

2. At each step of the algorithm, some queries are discarded, because they are more specific than other rewritings, even if they have not been explored yet. The question is whether this dynamic pruning of the search space keeps the completeness of the output. More generally, which properties have to be fulfilled by the operator to ensure the correctness of the algorithm and its termination for *fus* rules?

3. Finally, design a rewriting operator that fulfills the desired properties and leads to the effective computation of a sound and complete rewriting set.

With respect to the first question, we show that all sound and complete sets of rewritings, restricted to their most general elements, have the *same* cardinality, which is minimal with respect to the completeness property. Moreover, if we delete redundant atoms from the obtained CQs (which can be performed by a polynomial number of homomorphism tests for each query)[1], then we obtain a unique minimal sound and complete set of CQs of minimal size; uniqueness is of course up to a bijective variable renaming.

To answer the second question, we define several properties that a rewriting operator has to satisfy and show that these properties actually ensure the correctness of the algorithm and its termination for *fus* rules. In particular, we point out that the fact that a query may be removed from the rewriting set before being explored may prevent the completeness of the output, even if the rewriting operator is theoretically able to generate a complete output. The *prunability* of the rewriting operator ensures that this dynamic pruning can be safely performed. Briefly, this property holds if, for all queries $Q_1$ and $Q_2$, when $Q_1$ is more general than $Q_2$ then any one-step rewriting of $Q_2$ is less general than $Q_1$ itself or one of the one-step rewritings of $Q_1$; intuitively, this allows to discard the rewriting $Q_2$ even when its one-step rewritings have not been generated yet. Note that this kind of properties ties in with an issue raised in [ISG12] about the gap between theoretical completeness of some methods and the effective completeness of their implementation, this gap being mainly due to algorithmic optimizations (here the dynamic pruning).

[1] See e.g. [CM09], Section 2.6, on basic conceptual graphs. The algorithm can even by made linear, noticing that an atom needs to be considered only once.

Concerning the third question, we proceed in several steps. First, we rely on a specific unifier, called a *piece-unifier*, that was designed for backward chaining with conceptual graph rules (whose logical translation is exactly existential rules [SM96]). As in classical backward chaining, the rewriting process is based on a unification operation between the current query and a rule head. However, existential variables in rule heads induce a structure that has to be considered to keep soundness. Thus, instead of unifying a single atom of the query at once, our unifier processes a subset of atoms from the query. A *piece* is a minimal subset of atoms from the query that have to be erased together, hence the name piece-unifier. We present below a very simple example of piece unification (in particular, the head of the existential rule is restricted to a single atom).

**Example 2** *Let $R = \forall x\ (q(x) \rightarrow \exists y\ p(x, y))$ and the BCQ $Q = \exists u \exists v \exists w (p(u, v) \wedge p(w, v) \wedge r(u, w))$. Assume we want to unify the atom $p(u, v)$ from $Q$ with $p(x, y)$, for instance by a substitution $\{(u, x), (v, y)\}$.*[2] *Since $v$ is unified with the existential variable $y$, all other atoms from $Q$ containing $v$ must also be considered: indeed, simply rewriting $Q$ into $Q_1 = \exists w \exists x \exists y (q(x) \wedge p(w, y) \wedge r(x, w))$ would be unsound: intuitively, the fact that the atoms $p(u, v)$ and $p(w, v)$ in $Q$ share a variable would be lost in atoms $q(x)$ and $p(w, y)$; for instance $F = q(a) \wedge p(b, c) \wedge r(a, b)$ would answer $Q_1$ despite $Q$ being not entailed by $F$ and $R$. Thus, $p(u, v)$ and $p(w, v)$ have to be both unified with the head of $R$, for instance by means of the following substitution: $\mu = \{(u, x), (v, y), (w, x)\}$. $\{p(u, v), p(w, v)\}$ is called a piece. The corresponding rewriting of $Q$ is $\exists x (q(x) \wedge r(x, x))$.*

Piece-unifiers lead to a logically sound and complete rewriting method. As far as we know, it is the only method accepting any kind of existential rules, while staying in this fragment, i.e., without Skolemization of rule heads to replace existential variables with Skolem functions.

We show that the piece-based rewriting operator fulfills the desired properties ensuring the correctness of the generic algorithm, and its termination in the case of *fus* rules.

The next question was how to optimize the rewriting step. Indeed, the problem of deciding whether there is a piece-unifier between a query and a rule head is NP-complete and the number of piece-unifiers can be exponential in the size of the query. To cope with these sources of complexity, we consider so-called *single-piece* unifiers, which unify a single-piece of the query at once (like $\mu$ in Example 2). When, additionally, the head of a rule $R$ is restricted to an atom, which is a frequent case, each atom in a query $Q$ belongs to at most one piece with respect to $R$; then, the number of (most general) single-piece unifiers of $Q$ with the head of $R$ is bounded by the size of $Q$.

We show that the single-piece based rewriting operator is able to generate a sound and complete rewriting set. However, as pointed out in several examples, it is not prunable. Hence, single-piece unifiers have to be combined to recover prunability. We thus define the *aggregation* of single-piece unifiers and show that the corresponding rewriting operator fulfills all desired properties and generates fewer queries than the piece-based rewriting operator. Detailed algorithms are given and first experiments are reported.

**Paper organization.** Section 2 recalls some basic notions about the existential rule framework. Section 3 defines sound, complete and minimal sets of rewritings. In

[2] A substitution is given as a set of pairs, where a pair $(x, e)$ means that $x$ is substituted by $e$.

Section 4, the generic breadth-first algorithm is introduced and general properties of rewriting operators are studied. Section 5 presents the piece-based rewriting operator. In Section 6, we focus on exploiting single-piece unifiers and introduce the rewriting operator based on their aggregation. Finally, Section 7 is devoted to detailed algorithms and experiments, as well as to further work.

This is an extended version of papers by the same authors published at RR 2012 and RR 2013 (International Conference on Web Reasoning and Rule Systems).

# 2 Preliminaries

An *atom* is of the form $p(t_1, \ldots, t_k)$ where $p$ is a predicate with arity $k$, and the $t_i$ are terms, i.e., variables or constants. Given an atom or a set of atoms $A$, *vars*$(A)$, *consts*$(A)$ and *terms*$(A)$ denote its sets of variables, constants and terms, respectively. In all the examples in this paper, the terms are variables (denoted by $x$, $y$, $z$, etc.). $\models$ denotes the classical logical consequence. Two formulas $f_1$ and $f_2$ are said to be equivalent if $f_1 \models f_2$ and $f_2 \models f_1$.

A *fact* is an existentially closed conjunction of atoms.[3] A *conjunctive query* (CQ) is an existentially quantified conjunction of atoms. When it is a closed formula, it is called a *Boolean* CQ (BCQ). Hence, facts and BCQs have the same logical form. In the following, we will see them as sets of atoms. A union of conjunctive queries (UCQ) is a disjunction of CQs, which will see as a set of CQs.

Given sets of atoms $A$ and $B$, a *homomorphism* $h$ from $A$ to $B$ is a substitution of *vars*$(A)$ by *terms*$(B)$ such that $h(A) \subseteq B$. We say that $A$ is *mapped* to $B$ by $h$. If there is a homomorphism from $A$ to $B$, we say that $A$ is *more general* than $B$, which is denoted $A \geq B$.

Given a fact $F$ and a BCQ $Q$, the answer to $Q$ in $F$ is *positive* if $F \models Q$. It is well-known that $F \models Q$ if and only if there is a homomorphism from $Q$ to $F$. If $Q$ is a non-Boolean CQ, let $x_1 \ldots x_q$ be the free variables in $Q$. Then, a tuple of constants $(a_1 \ldots a_q)$ is an answer to $Q$ in $F$ if there is a homomorphism from $Q$ to $F$ that maps $x_i$ to $a_i$ for each $i$. In other words, $(a_1 \ldots a_q)$ is an answer to $Q$ in $F$ if and only if the answer to the BCQ obtained from $Q$ by substituting each $x_i$ with $a_i$ is positive.

In this paper, we consider only Boolean queries for simplicity reasons. This is not a restriction, since our mechanisms can actually process a CQ with free variables $x_1 \ldots x_q$ by translating it into a BCQ with an added atom $ans(x_1 \ldots x_q)$, where $ans$ is a special predicate not occurring in the knowledge base. Since $ans$ can never be erased by a rewriting step, the $x_i$ can only be substituted and will not "disappear". We can thus compute the rewriting set of a CQ as a Boolean CQ with a special $ans$ atom, then transform the rewritings into non-Boolean CQs by removing the $ans$ atom and consider its arguments as free variables. Note that our the generic algorithm can accept as input a union of conjunctive queries as well, since it works exactly in the same way if it takes as input a set of CQs instead of a single CQ.

**Definition 1 (Existential rule)** *An* existential rule *(or simply a* rule*) is a formula* $R = \forall \vec{x} \forall \vec{y} (B[\vec{x}, \vec{y}] \rightarrow \exists \vec{z} H[\vec{y}, \vec{z}])$*, where* $\vec{x}, \vec{y}$ *and* $\vec{z}$ *are tuple of variables,* $B =$ *body*$(R)$ *and* $H =$ *head*$(R)$ *are conjunctions of atoms, resp. called the* body *and the* head *of* $R$*. The* frontier *of* $R$*, denoted by* fr$(R)$*, is the set* vars$(B) \cap$ vars$(H) = \vec{y}$*. The set of* existential variables *in* $R$ *is the set* vars$(H) \setminus$ fr$(R) = \vec{z}$*.*

[3] We generalize the classical notion of a fact in order to take existential variables into account.

In the following, we omit quantifiers in rules and queries, as there is no ambiguity. For instance, the rule $R = \forall x\ (q(x) \rightarrow \exists y\ p(x, y))$ from Example 2 will be written $q(x) \rightarrow p(x, y)$.

A *knowledge base* (KB) $\mathcal{K} = (F, \mathcal{R})$ is composed of a fact $F$ and a finite set of existential rules $\mathcal{R}$. The *BCQ entailment problem* takes as input a KB $\mathcal{K} = (F, \mathcal{R})$ and a BCQ $Q$, and asks if $F, \mathcal{R} \models Q$ holds.

# 3 Desirable Properties of Rewriting Sets

Given a query $Q$ and a set of existential rules $\mathcal{R}$, rewriting techniques compute a set of queries $\mathcal{Q}$, which we call a *rewriting set* hereafter. It is generally desired that such a set satisfies at least three properties: *soundness*, *completeness* and *minimality*.

**Definition 2 (Sound and Complete set)** *Let $\mathcal{R}$ be a set of existential rules and $Q$ be a BCQ. Let $\mathcal{Q}$ be a set of BCQs. $\mathcal{Q}$ is said to be* sound *w.r.t. $Q$ and $\mathcal{R}$ if for all facts $F$, for all $Q' \in \mathcal{Q}$, if $Q'$ can be mapped to $F$ then $F, \mathcal{R} \models Q$. Reciprocally, $\mathcal{Q}$ is said to be* complete *w.r.t. $Q$ and $\mathcal{R}$ if for all facts $F$, if $F, \mathcal{R} \models Q$ then there is $Q' \in \mathcal{Q}$ such that $Q'$ can be mapped to $F$.*

We mentioned in the introduction that only the *most general elements* of a rewriting set need to be considered. Indeed, let $Q_1$ and $Q_2$ be two elements of a rewriting set such that $Q_1 \geq Q_2$. Then, for any fact $F$, the set of answers to $Q_2$ in $F$ is included in the set of answers to $Q_1$ in $F$. Hence, removing $Q_2$ will not undermine completeness (and it will not undermine soundness either). The output of a rewriting algorithm should thus be a minimal set of incomparable queries that "covers" the set of all the sound rewritings of the initial query.

**Definition 3 (Covering relation)** *Let $\mathcal{Q}_1$ and $\mathcal{Q}_2$ be two sets of BCQs. $\mathcal{Q}_1$* covers *$\mathcal{Q}_2$, which is denoted $\mathcal{Q}_1 \geq \mathcal{Q}_2$, if for all $Q_2 \in \mathcal{Q}_2$ there is $Q_1 \in \mathcal{Q}_1$ with $Q_1 \geq Q_2$.*

Note that covering can also be defined in terms of classical database query containment, i.e., $\mathcal{Q}_1$ covers $\mathcal{Q}_2$ if and only if the UCQ $\mathcal{Q}_2$ is included in the UCQ $\mathcal{Q}_1$.

**Definition 4 (Minimal set of BCQs, Cover)** *Let $\mathcal{Q}$ be a set of BCQs. $\mathcal{Q}$ is said to be* minimal *if there is no $Q \in \mathcal{Q}$ such that $(\mathcal{Q} \setminus \{Q\}) \geq \mathcal{Q}$. A* cover *of $\mathcal{Q}$ is a minimal set $\mathcal{Q}^c \subseteq \mathcal{Q}$ such that $\mathcal{Q}^c \geq \mathcal{Q}$.*

Since a cover is a minimal set, its elements are pairwise incomparable.

**Example 3** *Let $\mathcal{Q} = \{Q_1, \ldots, Q_6\}$ and consider the following preorder over $\mathcal{Q}$: $Q_1 \geq Q_2, Q_4, Q_5, Q_6$ ; $Q_2 \geq Q_1, Q_4, Q_5, Q_6$ ; $Q_3 \geq Q_4$ ; $Q_5 \geq Q_6$ (note that $Q_1$ and $Q_2$ are equivalent). There are two covers of $\mathcal{Q}$, namely $\{Q_1, Q_3\}$ and $\{Q_2, Q_3\}$. See Figure 2.*

A set of (sound) rewritings may have a finite cover even when it is infinite, as illustrated by Example 4.

**Example 4** *Let $Q = t(u), R_1 = t(x) \wedge p(x, y) \rightarrow r(y), R_2 = r(x) \wedge p(x, y) \rightarrow t(y)$. $R_1$ and $R_2$ have a head restricted to a single atom and no existential variable, hence the classical most general unifier can be used, which unifies the first atom in the query with the atom of a rule head. The rewriting set of $Q$ with $\{R_1, R_2\}$ is infinite. The*

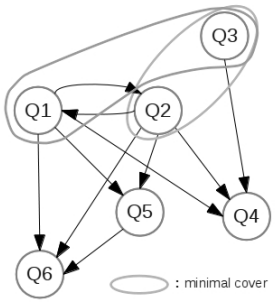


Figure 2: Cover (Example 3)

*first generated queries are the following (note that rule variables are renamed when needed):*
$Q_0 = t(u)$
$Q_1 = r(x) \wedge p(x, y)$ *// from* $Q_0$ *and* $R_2$ *with* $\{(u, y)\}$
$Q_2 = t(x_0) \wedge p(x_0, y_0) \wedge p(y_0, y)$ *// from* $Q_1$ *and* $R_1$ *with* $\{(x, y_0)\}$
$Q_3 = r(x_1) \wedge p(x_1, y_1) \wedge p(y_1, y_0) \wedge p(y_0, y)$ *// from* $Q_2$ *and* $R_2$ *with* $\{(x_0, y_1)\}$
$Q_4 = t(x_2) \wedge p(x_2, y_2) \wedge p(y_2, y_1) \wedge p(y_1, y_0) \wedge p(y_0, y)$ *// from* $Q_3$ *and* $R_1$
and so on ...
*However, the set of the most general rewritings is* $\{Q_0, Q_1\}$ *since any other query that can be obtained is more specific than* $Q_0$ *or* $Q_1$.

It can be easily checked that all covers of a given set have the same cardinality. We now prove that this property can be extended to the covers of all sound and complete finite rewriting sets of $Q$, irrespective of the rewriting technique used to compute these sets.

**Theorem 1** *Let $\mathcal{R}$ be a set of rules and $Q$ be a BCQ. Any finite cover of a sound and complete rewriting set of $Q$ with $\mathcal{R}$ is of minimal cardinality (among all sound and complete rewriting sets of $Q$).*

*Proof:* Let $\mathcal{Q}_1$ and $\mathcal{Q}_2$ be two arbitrary sound and complete rewriting sets of $Q$ with $\mathcal{R}$. Let $\mathcal{Q}_1^c$ (resp. $\mathcal{Q}_2^c$) be one of the finite covers of $\mathcal{Q}_1$ (resp. $\mathcal{Q}_2$). $\mathcal{Q}_1^c$ (resp. $\mathcal{Q}_2^c$) is also sound and complete, as well as smaller than or equal to $\mathcal{Q}_1$ (resp. $\mathcal{Q}_2$). We show that they have the same cardinality. Let $Q_1 \in \mathcal{Q}_1^c$. There exists $Q_2 \in \mathcal{Q}_2^c$ such that $Q_2 \geq Q_1$. If not, $Q$ would be entailed by $F = Q_1$ and $\mathcal{R}$ since $\mathcal{Q}_1^c$ is a sound rewriting set of $Q$ (and $Q_1$ maps to itself), but no elements of $\mathcal{Q}_2^c$ would map to $F$: thus, $\mathcal{Q}_2^c$ would not be complete. Similarly, there exists $Q_1' \in \mathcal{Q}_1^c$ such that $Q_1' \geq Q_2$. Then $Q_1' \geq Q_1$, which implies that $Q_1' = Q_1$ by assumption on $Q_1^c$. For all $Q_1 \in \mathcal{Q}_1^c$, there exists $Q_2 \in \mathcal{Q}_2^c$ such that $Q_2 \geq Q_1$ and $Q_1 \geq Q_2$. Such a $Q_2$ is unique: indeed, two such elements would be comparable for $\geq$, which is not possible by construction of $\mathcal{Q}_2^c$. The function associating $Q_2$ with $Q_1$ is thus a bijection from $\mathcal{Q}_1^c$ to $\mathcal{Q}_2^c$, which shows that these two sets have the same cardinality. □

Furthermore, the proof of the preceding theorem shows that, given any two sound and complete rewriting sets of $Q$, there is a bijection from any cover of the first set

to any cover of the second set such that two elements in relation by the bijection are equivalent. However, these elements are not necessarily isomorphic (i.e., equal up to a variable renaming) because they may contain redundancies. Consider the preorder induced by homomorphism on the set of all BCQs definable on some vocabulary. It is well-known that this preorder is such that any of its equivalence classes possesses a unique element of minimal size (up to isomorphism), called its *core* (notion introduced for graphs[4], but easily transferable to queries).

Every query can be transformed into its equivalent core by removing redundant atoms. We recall that a set of existential rules ensuring that a finite sound and complete set of most general rewritings exists for any query is called a finite unification set (*fus*).[5]

By the remark above and Theorem 1, we obtain:

**Corollary 2** *Let $\mathcal{R}$ be a fus and $Q$ be a BCQ. There is a unique finite sound and complete rewriting set of $Q$ with $\mathcal{R}$ that has both minimal cardinality and elements of minimal size.*

# 4 A Generic Rewriting Algorithm

We will now present a generic rewriting algorithm that takes as input a set of existential rules and a query, and as parameter a *rewriting operator*. The studied question is the following: which properties should this operator satisfy in order that the algorithm outputs a sound, complete, finite and minimal set?

## 4.1 Rewriting Algorithm

**Definition 5 (Rewriting operator)** *A* rewriting operator $\texttt{rew}$ *is a function which takes as input a BCQ $Q$ and a set of rules $\mathcal{R}$ and outputs a set of BCQs denoted by $\texttt{rew}(Q, \mathcal{R})$.*

Since the elements of $\texttt{rew}(Q, \mathcal{R})$ are BCQs, it is possible to apply further steps of rewriting to them. This naturally leads to the notions of $k$-rewriting and $k$-saturation.

**Definition 6 ($k$-rewriting)** *Let $Q$ be a conjunctive query, $\mathcal{R}$ be a set of rules and $\texttt{rew}$ be a rewriting operator. A* 1-rewriting *of $Q$ (w.r.t. $\texttt{rew}$ and $\mathcal{R}$) is an element of $\texttt{rew}(Q, \mathcal{R})$. A* $k$-rewriting *of $Q$, for $k > 1$, (w.r.t. $\texttt{rew}$ and $\mathcal{R}$) is a* 1*-rewriting of a $(k-1)$-rewriting of $Q$.*

The term $k$-saturation is convenient to name the set of queries that can be obtained in at most $k$ rewriting steps.

**Definition 7 ($k$-saturation)** *Let $Q$ be a BCQ, $\mathcal{R}$ be a set of rules and $\texttt{rew}$ be a rewriting operator. We denote the set of $k$-rewritings of $Q$ by $\texttt{rew}_k(Q, \mathcal{R})$. We call* $k$-saturation*, and denote by $W_k(Q, \mathcal{R})$, the set of $i$-rewritings of $Q$ for all $i \leq k$. We denote $W_\infty(Q, \mathcal{R}) = \bigcup_{k \in \mathbb{N}} W_k(Q, \mathcal{R})$.*

In the following, we extend the notations $\texttt{rew}$, $\texttt{rew}_k$ and $W_k$ to a set of BCQs $\mathcal{Q}$ instead of a single BCQ $Q$: $\texttt{rew}(\mathcal{Q}, \mathcal{R}) = \bigcup_{Q \in \mathcal{Q}} \texttt{rew}(Q, \mathcal{R})$, $\texttt{rew}_k(\mathcal{Q}, \mathcal{R}) = \bigcup_{Q \in \mathcal{Q}} \texttt{rew}_k(Q, \mathcal{R})$ and $W_k(\mathcal{Q}, \mathcal{R}) = \bigcup_{i \leq k} \texttt{rew}_i(\mathcal{Q}, \mathcal{R})$.

[4] See for instance [HN92], where the notion of a core is traced back to the late sixties.

[5] The notion of a finite unification set was first introduced in [BLMS09] and defined with respect to piece-unifiers. However, since piece-unifiers provide a sound and complete rewriting operator (see Section 5) and all the covers of a given set have the same cardinality, the two definitions are equivalent.

Algorithm 1 performs a breadth-first exploration of the rewriting space of a given query.[6] At each step, only the most general elements are kept thanks to a covering function, denoted by `cover` , that computes a cover of a given set.

**Algorithm 1:** A GENERIC REWRITING ALGORITHM

**Data**: A set of rules $\mathcal{R}$, a BCQ $Q$
**Access**: A rewriting operator `rew` , a covering function `cover`
**Result**: A cover of the set of all the rewritings of $Q$
$\mathcal{Q}_F \leftarrow \{Q\}$; *// resulting set*
$\mathcal{Q}_E \leftarrow \{Q\}$; *// queries to be explored*
**while** $\mathcal{Q}_E \neq \emptyset$ **do**
  $\mathcal{Q}_C \leftarrow \texttt{cover}(\mathcal{Q}_F \cup \texttt{rew}(\mathcal{Q}_E, \mathcal{R}))$; *// update cover*
  $\mathcal{Q}_E \leftarrow \mathcal{Q}_C \backslash \mathcal{Q}_F$; *// select unexplored queries*
  $\mathcal{Q}_F \leftarrow \mathcal{Q}_C$;
**return** $\mathcal{Q}_F$

For termination reasons (see the proof of Property 6), already explored queries are preferred to non-explored queries in the computation of the cover. More precisely, if both $\mathcal{Q}_c \cup \{q\}$ and $\mathcal{Q}_c \cup \{q'\}$ are covers of $\mathcal{Q}_F \cup \texttt{rew}(\mathcal{Q}_E, \mathcal{R})$, with $q$ and $q'$ homomorphically equivalent and $\{q\}$ belongs to $\mathcal{Q}_F$, then `cover` does not output $\mathcal{Q}_c \cup \{q'\}$. If `rew` fulfills some good properties (specified below), then, after the $i^{th}$ iteration of the while loop, the $i$-saturation of $Q$ (with respect to $\mathcal{R}$ and `rew`) is covered by $\mathcal{Q}_F$, while $\mathcal{Q}_E$ contains the queries that remain to be explored.

In the remainder of this section, we study the conditions that a rewriting operator must meet in order that: (i) the algorithm halts and outputs a cover of all the rewritings that can be obtained with this rewriting operator, provided that such a finite cover exists; (ii) the output cover is sound and complete.

## 4.2 Correctness and Termination of the Algorithm

We now define a property on the rewriting operator, called *prunability*. This property is sufficient to ensure that Algorithm 1 outputs a cover of $W_\infty(Q, \mathcal{R})$. Intuitively, if an operator is prunable then, for every $Q_1$ more general than $Q_2$, the one-step rewritings of $Q_2$ are covered by the one-step rewritings of $Q_1$ or by $Q_1$ itself. It follows that all the rewritings of $Q_2$ are covered by $Q_1$ and its rewritings. Hence, $Q_2$ can be safely removed from the current rewriting set.

**Definition 8 (Prunable)** *A rewriting operator* `rew` *is said to be* prunable *if for any set of rules $\mathcal{R}$ and for all BCQs $Q_1, Q_2, Q'_2$ such that $Q_1 \geq Q_2$, $Q'_2 \in \texttt{rew}(Q_2, \mathcal{R})$ and $Q_1 \not\geq Q'_2$, there is $Q'_1 \in \texttt{rew}(Q_1, \mathcal{R})$ such that $Q'_1 \geq Q'_2$.*

The following lemma states that this can be generalized to $k$-rewritings for any $k$.

**Lemma 3** *Let* `rew` *be a prunable rewriting operator, and let $\mathcal{Q}_1$ and $\mathcal{Q}_2$ be two sets of BCQs. If $\mathcal{Q}_1 \geq \mathcal{Q}_2$, then $W_\infty(\mathcal{Q}_1, \mathcal{R}) \geq W_\infty(\mathcal{Q}_2, \mathcal{R})$.*

*Proof:* We prove by induction on $i$ that $W_i(\mathcal{Q}_1, \mathcal{R}) \geq \texttt{rew}_i(\mathcal{Q}_2, \mathcal{R})$.
For $i = 0$, $W_0(\mathcal{Q}_1, \mathcal{R}) = \mathcal{Q}_1 \geq \mathcal{Q}_2 = \texttt{rew}_0(\mathcal{Q}_2, \mathcal{R})$.
For $i > 0$, for any $Q_2 \in \texttt{rew}_i(\mathcal{Q}_2, \mathcal{R})$, there is $Q'_2 \in \texttt{rew}_{i-1}(\mathcal{Q}_2, \mathcal{R})$ such that $Q_2 \in \texttt{rew}(Q'_2, \mathcal{R})$. By induction hypothesis, there is $Q'_1 \in W_{i-1}(\mathcal{Q}_1, \mathcal{R})$ such that

[6] Note that a depth-first exploration would not ensure termination for *fus* rules.

$Q'_1 \geq Q'_2$. $\texttt{rew}$ is prunable, thus either $Q'_1 \geq Q_2$ or there is $Q_1 \in \texttt{rew}(Q'_1, \mathcal{R})$ such that $Q_1 \geq Q_2$. Since $W_{i-1}(\mathcal{Q}_1, \mathcal{R})$ and $\texttt{rew}(Q'_1, \mathcal{R})$ are both included in $W_i(\mathcal{Q}_1, \mathcal{R})$, we can conclude. □

This lemma would not be sufficient to prove the correctness of Algorithm 1, as will be discussed in Section 6.1. We need a stronger version, which checks that a query whose 1-rewritings are covered needs not to be explored.

**Lemma 4** *Let $\texttt{rew}$ be a prunable rewriting operator, and let $\mathcal{Q}_1$ and $\mathcal{Q}_2$ be two sets of BCQs. If $(\mathcal{Q}_1 \cup \mathcal{Q}_2) \geq \texttt{rew}(\mathcal{Q}_1, \mathcal{R})$, then $(\mathcal{Q}_1 \cup W_\infty(\mathcal{Q}_2, \mathcal{R})) \geq W_\infty(\mathcal{Q}_1 \cup \mathcal{Q}_2, \mathcal{R})$.*

*Proof:* We prove by induction on $i$ that $\mathcal{Q}_1 \cup W_i(\mathcal{Q}_2, \mathcal{R}) \geq \texttt{rew}_i(\mathcal{Q}_1 \cup \mathcal{Q}_2, \mathcal{R})$.
For $i = 0$, $\texttt{rew}_0(\mathcal{Q}_1 \cup \mathcal{Q}_2, \mathcal{R}) = \mathcal{Q}_1 \cup \mathcal{Q}_2 = \mathcal{Q}_1 \cup W_0(\mathcal{Q}_2, \mathcal{R})$.
For $i > 0$, for any $Q_i \in \texttt{rew}_i(\mathcal{Q}_1 \cup \mathcal{Q}_2, \mathcal{R})$, there is $Q_{i-1} \in \texttt{rew}_{i-1}(\mathcal{Q}_1 \cup \mathcal{Q}_2, \mathcal{R})$ such that $Q_i \in \texttt{rew}(Q_{i-1}, \mathcal{R})$. By induction hypothesis, there is $Q'_{i-1} \in \mathcal{Q}_1 \cup W_{i-1}(\mathcal{Q}_2, \mathcal{R})$ such that $Q'_{i-1} \geq Q_{i-1}$. Since $\texttt{rew}$ is prunable, either $Q'_{i-1} \geq Q_i$ or there is $Q'_i \in \texttt{rew}(Q'_{i-1}, \mathcal{R})$ such that $Q'_i \geq Q_i$. Then, there are two possibilities:

- either $Q'_{i-1} \in \mathcal{Q}_1$: since $\mathcal{Q}_1 \cup \mathcal{Q}_2 \geq \texttt{rew}(\mathcal{Q}_1, \mathcal{R})$, we have $\mathcal{Q}_1 \cup \mathcal{Q}_2 \geq \{Q'_i\}$ and so $\mathcal{Q}_1 \cup W_i(\mathcal{Q}_2, \mathcal{R}) \geq \{Q'_i\}$.
- or $Q'_{i-1} \in W_{i-1}(\mathcal{Q}_2, \mathcal{R})$: then $Q'_i \in W_i(\mathcal{Q}_2, \mathcal{R})$.

□

Finally, the correctness of Algorithm 1 is based on the following loop invariants.

**Property 5 (Invariants of Algorithm 1)** *Let $\texttt{rew}$ be a rewriting operator. After each iteration of the while loop of Algorithm 1, the following properties hold:*

1. $\mathcal{Q}_E \subseteq \mathcal{Q}_F \subseteq W_\infty(Q, \mathcal{R})$;
2. $\mathcal{Q}_F \geq \texttt{rew}(\mathcal{Q}_F \setminus \mathcal{Q}_E, \mathcal{R})$;
3. *if $\texttt{rew}$ is prunable then* $(\mathcal{Q}_F \cup W_\infty(\mathcal{Q}_E, \mathcal{R})) \geq W_\infty(Q, \mathcal{R})$;
4. *for all distinct $Q, Q' \in \mathcal{Q}_F$, $Q \not\geq Q'$ and $Q' \not\geq Q$.*

*Proof:* Invariants are proved by induction on the number of iterations of the while loop. Below $\mathcal{Q}_F^i$ and $\mathcal{Q}_E^i$ denote the value of $\mathcal{Q}_F$ and $\mathcal{Q}_E$ after $i$ iterations.

*Invariant 1:* $\mathcal{Q}_E \subseteq \mathcal{Q}_F \subseteq W_\infty(Q, \mathcal{R})$.

*basis:* $\mathcal{Q}_E^0 = \mathcal{Q}_F^0 = \{Q\} = W_0(Q, \mathcal{R}) \subseteq W_\infty(Q, \mathcal{R})$.

*induction step:* by construction, $\mathcal{Q}_E^i \subseteq \mathcal{Q}_F^i$ and $\mathcal{Q}_F^i \subseteq \mathcal{Q}_F^{i-1} \cup \texttt{rew}(\mathcal{Q}_E^{i-1}, \mathcal{R})$. For any $Q' \in \mathcal{Q}_F^i$ we have: either $Q' \in \mathcal{Q}_F^{i-1}$ and then by induction hypothesis $Q' \in W_\infty(Q, \mathcal{R})$; or $Q' \in \texttt{rew}(\mathcal{Q}_E^{i-1}, \mathcal{R})$ and then by induction hypothesis we have $\mathcal{Q}_E^{i-1} \subseteq W_\infty(Q, \mathcal{R})$, which implies $Q' \in W_\infty(Q, \mathcal{R})$.

*Invariant 2:* $\mathcal{Q}_F \geq \texttt{rew}(\mathcal{Q}_F \setminus \mathcal{Q}_E, \mathcal{R})$.

*basis:* $\texttt{rew}(\mathcal{Q}_F^0 \setminus \mathcal{Q}_E^0, \mathcal{R}) = \texttt{rew}(\emptyset, \mathcal{R}) = \emptyset$ and any set covers it.

*induction step:* by construction, $\mathcal{Q}_F^i \geq \mathcal{Q}_F^{i-1} \cup \texttt{rew}(\mathcal{Q}_E^{i-1}, \mathcal{R})$; since by induction hypothesis $\mathcal{Q}_F^{i-1} \geq \texttt{rew}(\mathcal{Q}_F^{i-1} \setminus \mathcal{Q}_E^{i-1}, \mathcal{R})$, we have $\mathcal{Q}_F^i \geq \texttt{rew}(\mathcal{Q}_F^{i-1} \setminus \mathcal{Q}_E^{i-1}, \mathcal{R}) \cup \texttt{rew}(\mathcal{Q}_E^{i-1}, \mathcal{R}) = \texttt{rew}(\mathcal{Q}_F^{i-1}, \mathcal{R})$. Furthermore, by construction, $\mathcal{Q}_E^i = \mathcal{Q}_F^i \setminus \mathcal{Q}_F^{i-1}$; thus $\mathcal{Q}_F^i \setminus \mathcal{Q}_E^i \subseteq \mathcal{Q}_F^{i-1}$ and so $\texttt{rew}(\mathcal{Q}_F^i \setminus \mathcal{Q}_E^i, \mathcal{R}) \subseteq \texttt{rew}(\mathcal{Q}_F^{i-1}, \mathcal{R})$. Thus $\mathcal{Q}_F^i \geq \texttt{rew}(\mathcal{Q}_F^i \setminus \mathcal{Q}_E^i, \mathcal{R})$.

*Invariant 3:* if $\texttt{rew}$ is prunable then $(\mathcal{Q}_F \cup W_\infty(\mathcal{Q}_E, \mathcal{R})) \geq W_\infty(Q, \mathcal{R})$.

*basis:* $(\mathcal{Q}_F^0 \cup W_\infty(\mathcal{Q}_E^0, \mathcal{R})) = (\{Q\} \cup W_\infty(\{Q\}, \mathcal{R})) = W_\infty(Q, \mathcal{R})$.

*induction step:* we first show that *(i):* $(\mathcal{Q}_F^i \cup W_\infty(\mathcal{Q}_E^i, \mathcal{R})) \geq W_\infty(\mathcal{Q}_F^i, \mathcal{R})$, then we prove by induction that *(ii):* $W_\infty(\mathcal{Q}_F^i, \mathcal{R}) \geq W_\infty(Q, \mathcal{R})$:

- *(i)* by construction $\mathcal{Q}_E^i \subseteq \mathcal{Q}_F^i$, thus $(\mathcal{Q}_F^i \setminus \mathcal{Q}_E^i) \cup \mathcal{Q}_E^i = \mathcal{Q}_F^i$, and by Invariant 2, we have $(\mathcal{Q}_F^i \setminus \mathcal{Q}_E^i) \cup \mathcal{Q}_E^i \geq \texttt{rew}(\mathcal{Q}_F^i \setminus \mathcal{Q}_E^i, \mathcal{R})$. Lemma 4 then entails that $((\mathcal{Q}_F^i \setminus \mathcal{Q}_E^i) \cup W_\infty(\mathcal{Q}_E^i, \mathcal{R})) \geq W_\infty((\mathcal{Q}_F^i \setminus \mathcal{Q}_E^i) \cup \mathcal{Q}_E^i, \mathcal{R})$ and we can conclude since $\mathcal{Q}_F^i = (\mathcal{Q}_F^i \setminus \mathcal{Q}_E^i) \cup \mathcal{Q}_E^i$.
- *(ii)* by construction, we have $\mathcal{Q}_F^i \geq \mathcal{Q}_F^{i-1} \cup \texttt{rew}(\mathcal{Q}_E^{i-1}, \mathcal{R})$; so, by Lemma 3, we have $W_\infty(\mathcal{Q}_F^i, \mathcal{R}) \geq W_\infty(\mathcal{Q}_F^{i-1} \cup \texttt{rew}(\mathcal{Q}_E^{i-1}, \mathcal{R}), \mathcal{R}) = W_\infty(\mathcal{Q}_F^{i-1}, \mathcal{R}) \cup W_\infty(\texttt{rew}(\mathcal{Q}_E^{i-1}, \mathcal{R}), \mathcal{R})$. Moreover, $\mathcal{Q}_E^{i-1} \subseteq \mathcal{Q}_F^{i-1} \subseteq W_\infty(\mathcal{Q}_F^{i-1}, \mathcal{R})$, thus $W_\infty(\mathcal{Q}_F^i, \mathcal{R}) \geq \mathcal{Q}_F^{i-1} \cup \mathcal{Q}_E^{i-1} \cup W_\infty(\texttt{rew}(\mathcal{Q}_E^{i-1}, \mathcal{R}), \mathcal{R}) = \mathcal{Q}_F^{i-1} \cup W_\infty(\mathcal{Q}_E^{i-1}, \mathcal{R})$. Using *(i)*, we have $W_\infty(\mathcal{Q}_F^i, \mathcal{R}) \geq W_\infty(\mathcal{Q}_F^{i-1}, \mathcal{R})$ and conclude by induction hypothesis.

*Invariant 4:* for all distinct $Q, Q' \in \mathcal{Q}_F$, $Q \not\geq Q'$ and $Q' \not\geq Q$. Trivially satisfied thanks to the properties of $\texttt{cover}$ .

□

The next property states that if $\texttt{rew}$ is prunable then Algorithm 1 halts for each case where $W_\infty(Q, \mathcal{R})$ has a finite cover.

**Property 6** *Let $\texttt{rew}$ be a rewriting operator, $\mathcal{R}$ be a set of rules and $Q$ be a BCQ. If $W_\infty(Q, \mathcal{R})$ has a finite cover and $\texttt{rew}$ is prunable then Algorithm 1 halts.*

*Proof:* Let $\mathcal{Q}$ be a finite cover of $W_\infty(Q, \mathcal{R})$ and let $m$ be the largest $k$ for a $k$-rewriting in $\mathcal{Q}$.

We thus have $W_m(Q, \mathcal{R}) \geq \mathcal{Q} \geq W_\infty(Q, \mathcal{R})$. Since the operator is prunable, we have $\mathcal{Q}_F^i \geq W_i(Q, \mathcal{R})$ for all $i \geq 0$ (proved with a straightforward induction on $i$). Thus $\mathcal{Q}_F^m \geq W_\infty(Q, \mathcal{R})$. Thus, $\texttt{rew}(\mathcal{Q}_E^m, \mathcal{R})$ is covered by $Q_F^m$, and since already explored queries are taken first for the computation of a cover, we have that $\mathcal{Q}_E^{m+1} = \emptyset$. Hence Algorithm 1 halts. □

**Theorem 7** *Let $\texttt{rew}$ be a rewriting operator, $\mathcal{R}$ be a set of rules and $Q$ be a BCQ. If $W_\infty(Q, \mathcal{R})$ has a finite cover and $\texttt{rew}$ is prunable then Algorithm 1 outputs this cover (up to query equivalence).*

*Proof:* By Property 6, Algorithm 1 halts. By Invariant 3 from Property 5, $(\mathcal{Q}_F^f \cup W_\infty(\mathcal{Q}_E^f, \mathcal{R})) \geq W_\infty(Q, \mathcal{R})$ where $Q_F^f$ and $Q_E^f$ denote the final values of $Q_F$ and $Q_E$ in Algorithm 1. Since $Q_E^f = \emptyset$ when Algorithm 1 halts, we have $\mathcal{Q}_F^f \geq W_\infty(Q, \mathcal{R})$. Thanks to Invariants 1 and 4 from Property 5 we conclude that $\mathcal{Q}_F^f$ is a cover of $W_\infty(Q, \mathcal{R})$. □

## 4.3 Preserving Soundness and Completeness

We consider two further properties of a rewriting operator, namely soundness and completeness, with the aim of ensuring the soundness and completeness of the obtained rewriting set within the meaning of Definition 2.

**Definition 9 (Soundness/completeness of a rewriting operator)** *Let* `rew` *be a rewriting operator.* `rew` *is sound if for any set of rules $\mathcal{R}$, for any BCQ $Q$, for any $Q' \in$* `rew`$(Q, \mathcal{R})$*, for any fact $F$, $F \models Q'$ implies that $F, \mathcal{R} \models Q$.* `rew` *is complete if for any set of rules $\mathcal{R}$, for any BCQ $Q$, for any fact $F$ such that $F, \mathcal{R} \models Q$, there exists $Q' \in W_\infty(Q, \mathcal{R})$ such that $F \models Q'$.*

**Property 8** *If* `rew` *is sound, then the output of Algorithm 1 is a sound rewriting set of $Q$ and $\mathcal{R}$.*

*Proof:* Direct consequence of Invariant 1 from Property 5. □

Perhaps surprisingly, the completeness of the rewriting operator is not sufficient to ensure the completeness of the output rewriting set. Examples are provided in Section 6.1. This is due to the dynamic pruning performed at each step of Algorithm 1. Therefore the prunability of the operator is also required.

**Property 9** *If* `rew` *is prunable and complete, then the output of Algorithm 1 is a complete rewriting set of $Q$ and $\mathcal{R}$.*

*Proof:* Algorithm 1 returns $Q_F$ when $Q_E$ is empty. By Invariant 3 of Property 5, we know that $(\mathcal{Q}_F \cup W_\infty(\emptyset, \mathcal{R})) \geq W_\infty(Q, \mathcal{R})$. Since $W_\infty(\emptyset, \mathcal{R})) = \emptyset$, we obtain that $\mathcal{Q}_F \geq W_\infty(Q, \mathcal{R})$. □

Finally, as stated by the next theorem, when the rewriting operator is sound, complete and prunable, Algorithm 1 is correct and terminates for any finite unification set of rules. We remind that expressive classes of fus rules are known (see the introduction). In particular, the main members of DL-Lite family are generalized by the simple class of linear existential rules. See also Section 7 for examples of such ontologies.

**Theorem 10** *If* `rew` *is a sound, complete and prunable operator, and $\mathcal{R}$ is a finite unification set of rules, then for any BCQ $Q$, Algorithm 1 outputs a minimal (finite) sound and complete rewriting set of $Q$ with $\mathcal{R}$.*

*Proof:* If $\mathcal{R}$ is a fus and `rew` is a sound and complete operator then $W_\infty(Q, \mathcal{R})$ has a finite cover. The claim then follows from Properties 8 and 9 and Theorem 7. □

# 5 Piece-Based Rewriting

As mentioned in the introduction (and illustrated in Example 2), existential variables in rule heads induce a structure that has to be taken into account in the rewriting mechanism. Hence the classical notion of a unifier is replaced by that of a piece-unifier [BLMS11]. A piece-unifier "unifies" a subset $Q'$ of $Q$ with a subset $H'$ of $\mathit{head}(R)$, in the sense that the associated substitution $u$ is such that $u(Q') = u(H')$. Given a piece-unifier, $Q$ is partitioned into "pieces", which are minimal subsets of atoms that must be processed together. More specifically, the *cutpoints* are the variables from $Q'$ that are not unified with existential variables from $H'$ (i.e., they are unified with frontier variables or constants); then a *piece* in $Q$ is a minimal non-empty subset of atoms "glued" by variables other than cutpoints, i.e., connected by a path of variables that are not cutpoints. We recall below the definition of pieces given in [BLMS11] (where $T$ corresponds to the set of cutpoints).

**Definition 10 (Piece)** *[BLMS11] Let $A$ be a set of atoms and $T \subseteq \mathit{vars}(A)$. A* piece *of $A$ according to $T$ is a minimal non-empty subset $P$ of $A$ such that, for all $a$ and $a'$ in $A$, if $a \in P$ and $(\mathit{vars}(a) \cap \mathit{vars}(a')) \not\subseteq T$, then $a' \in P$.*

In this paper, we give a definition of a piece-unifier based on partitions rather than substitutions, which simplifies subsequent proofs. For any substitution $u$ from a set of variables $E_1$ to a set of terms $E_2$ associated with a piece-unifier, it holds that $E_1 \cap E_2 = \emptyset$. Thus, $u$ can be associated with a *partition* $P_u$ of $E_1 \cup E_2$ such that two terms are in the same class of $P_u$ if and only if they are merged by $u$; more specifically, we consider the equivalence classes of the symmetric, reflexive and transitive closure of the following relation $\sim$: $t \sim t'$ if $u(t) = t'$. Conversely, given a partition on a set of terms $E$, such that no class contains two constants, we can consider a substitution $u$ obtained by selecting an element of each class with priority given to constants: if $\{e_1 \dots e_k\}$ is a class in the partition and $e_i$ is a selected element, then for all $e_j$ with $1 \leq j \neq i \leq k$, we set $u(e_j) = e_i$. If we consider a total order on terms, such that constants are smaller than variables, then a unique substitution is obtained by taking the smallest element in each class. We call *admissible partition* a partition such that no class contains two constants.

The set of all partitions over a given set is structured in a *lattice* by the "*finer than*" relation (given two partitions $P_1$ and $P_2$, $P_1$ is finer than $P_2$, denoted by $P_1 \geq P_2$, if every class of $P_1$ is included in a class of $P_2$).[7] The *join* of several partitions is the partition obtained by making the union of their non-disjoint classes. The join of two admissible partitions may be a non-admissible partition. We say that several admissible partitions are *compatible* if their join is an admissible partition. Note that if the concerned partitions are relative to the same set $E$, then their join is their *greatest lower bound* in the partition lattice of $E$.

The following property makes a link between comparable partitions and comparable substitutions.

**Property 11** *Let $P_1$ and $P_2$ be two admissible partitions over the same set such that $P_1 \geq P_2$, with associated substitutions $u_1$ and $u_2$ respectively. Then there is a substitution $s$ such that $u_2 = s \circ u_1$ (i.e., $u_1$ is "more general" than $u_2$).*

*Proof:* The substitution $s$ is built as follows: for any class $C_i \in P_1$, let $C_j \in P_2$ be the class such that $C_i \subseteq C_j$. Let $e_i$ (resp. $e_j$) be the selected element in $C_i$ (resp. $C_j$); if $e_i \neq e_j$ (in this case, $e_i$ is necessarily a variable), then $s(e_i) = e_j$. It can be immediately checked that $u_2 = s \circ u_1$. □

In the following definition of a piece-unifier, we assume that $Q$ and $R$ have disjoint sets of variables. Given $Q' \subseteq Q$, we call *separating variables* from $Q'$, and denote by $sep(Q')$, the variables occurring in both $Q'$ and $(Q \setminus Q')$: $sep(Q') = vars(Q') \cap vars(Q \setminus Q')$.

**Definition 11 (Piece-Unifier, Cutpoint)** *A* piece-unifier *of $Q$ with $R$ is a triple $\mu = (Q', H', P_u)$, where $Q' \neq \emptyset$, $Q' \subseteq Q$, $H' \subseteq head(R)$ and $P_u$ is a partition on $terms(Q') \cup terms(H')$ satisfying the following three conditions:*

1. *$P_u$ is admissible;*
2. *if a class in $P_u$ contains an existential variable (from $H'$) then the other terms in the class are non-separating variables from $Q'$;*
3. *$u(H') = u(Q')$, where $u$ is a substitution associated with $P_u$.*

[7] Usually, the notation $\leq$ is used to denote the relation "finer than". We adopt the converse convention, which is more in line with substitutions and the $\geq$ preorder on CQs.

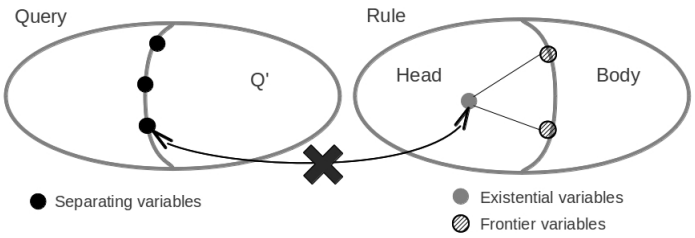


Figure 3: Piece-unifier

*The* cutpoints *of $\mu$, denoted by $cutp(\mu)$, are the variables from $Q'$ that are not unified with existential variables from $H'$ (i.e., they are unified with frontier variables or constants):* $cutp(\mu) = \{x \in vars(Q') \mid u(x) \in fr(R) \cup consts(Q') \cup consts(H')\}$.

Condition 2 in the piece-unifier definition ensures that a separating variable in $Q'$ is necessarily a cutpoint. It follows that $Q'$ is composed of pieces: indeed, an existential variable from $H'$ is necessarily unified with a non-separating variable from $Q'$, say $x$, which ensures that all atoms from $Q'$ in which $x$ occurs are also part of $Q'$. Figure 5 illustrates these notions.

We provide below some examples of piece-unifiers.

**Example 5** *Let $R = q(x) \rightarrow p(x, y)$ and $Q = p(u, v) \wedge p(w, v) \wedge p(w, t) \wedge r(u, w)$. Let $H' = \{p(x, y)\}$. They are three piece-unifiers of $Q$ with $R$:*
*$\mu_1 = (Q'_1, H', P^1_u)$ with $Q'_1 = \{p(u, v), p(w, v)\}$ and $P^1_u = \{\{x, u, w\}, \{y, v\}\}$*
*$\mu_2 = (Q'_2, H', P^2_u)$ with $Q'_2 = \{p(w, t)\}$ and $P^2_u = \{\{x, w\}, \{y, t\}\}$*
*$\mu_3 = (Q'_3, H', P^3_u)$ with $Q'_3 = \{p(u, v), p(w, v), p(w, t)\}$ and $P^3_u = \{\{x, u, w\}, \{y, v, t\}\}$*
*Note that $Q'_1$ and $Q'_2$ are each composed of a single piece; $Q'_3 = Q'_1 \cup Q'_2$ and $P^3_u$ is the join of $P^1_u$ and $P^2_u$.*

In the previous example, $R$ has an atomic head, thus a piece-unifier of $Q'$ with $R$ actually unifies the atoms from $Q'$ and the head of $R$ into a single atom. In the general case, a piece-unifier unifies $Q'$ and a subset $H'$ of $head(R)$ into a set of atoms, as illustrated by the next example.

**Example 6** *Let $R = q(x) \rightarrow p(x, y) \wedge p(y, z) \wedge p(z, t) \wedge r(y)$ and $Q = p(u, v) \wedge p(v, w) \wedge r(u)$. A piece-unifier of $Q$ with $R$ is $\mu_1 = (Q'_1, H'_1, P^1_u)$ with $Q'_1 = \{p(u, v), p(v, w)\}$, $H'_1 = \{p(x, y), p(y, z)\}$ and $P^1_u = \{\{x, u\}, \{v, y\}, \{w, z\}\}$. Another piece-unifier is $\mu_2 = (Q'_2, H'_2, P^2_u)$ with $Q'_2 = Q$, $H'_2 = \{p(y, z), p(z, t), r(y)\}$ and $P^2_u = \{\{u, y\}, \{v, z\}, \{w, t\}\}$.*
*Note that $\mu_3 = (Q'_3, H'_3, P^3_u)$ with $Q'_3 = \{p(u, v)\}$, $H'_3 = \{p(x, y)\}$ and $P^3_u = \{\{x, u\}, \{v, y\}\}$ is not a piece-unifier because the second condition in the definition of piece-unifier is not fulfilled: $v$ is a separating variable and is matched with the existential variable $y$.*

Then, the notions of a one-step rewriting according to a piece-unifier, and of a rewriting obtained by a sequence of one-step rewritings, are defined in the natural way.

**Definition 12 (One-step Piece-Rewriting)** *Given a piece-unifier $\mu = (Q', H', P_u)$ of $Q$ with $R$, the* one-step piece-rewriting *of $Q$ according to $\mu$, denoted by $\beta(Q, R, \mu)$, is the BCQ $u(\mathit{body}(R)) \cup u(Q \setminus Q')$, where $u$ is a substitution associated with $P_u$.*

We thus define inductively a *$k$-step piece-rewriting* as a $(k-1)$-step piece rewriting of a one-step piece-rewriting. For any $k$, a $k$-step piece-rewriting of $Q$ is a *piece-rewriting* of $Q$.

The next theorem states that piece-based rewriting is logically sound and complete.

**Theorem 12 ([SM96, BLMS11])** *Let $\mathcal{K} = (F, \mathcal{R})$ be a KB and $Q$ be a BCQ. Then $F, \mathcal{R} \models Q$ iff there is a piece-rewriting $Q'$ of $Q$ such that $Q' \geq F$.*

It follows from Theorem 12 that a sound and complete rewriting operator can be based on piece-unifiers: we call *piece-based rewriting operator*, the rewriting operator that, given $Q$ and $\mathcal{R}$, outputs all the one-step piece-rewritings of $Q$ according to a piece-unifier of $Q$ with $R \in \mathcal{R}$. We denote it by $\beta(Q, \mathcal{R})$.

Actually, as detailed hereafter, only most general piece-unifiers are to be considered, since the other piece-unifiers produce more specific queries.

**Definition 13 (Most General Piece-Unifier)** *Given two piece-unifiers defined on the same subsets of a query and a rule head, $\mu_1 = (Q', H', P_u^1)$ and $\mu_2 = (Q', H', P_u^2)$, we say that $\mu_1$ is* more general than *$\mu_2$ (notation $\mu_1 \geq \mu_2$) if $P_u^1$ is finer than $P_u^2$ (i.e., $P_u^1 \geq P_u^2$). A piece-unifier $\mu = (Q', H', P_u)$ is called a* most general *piece-unifier if it is more general than all the piece-unifiers on $Q'$ and $H'$.*

**Property 13** *Let $\mu_1$ and $\mu_2$ be two piece-unifiers with $\mu_1 \geq \mu_2$. Then $\mu_1$ and $\mu_2$ have the same pieces.*

*Proof:* $\mu_1$ and $\mu_2$ have the same pieces iff they have the same cutpoints. It holds that $\mathit{cutp}(\mu_1) \subseteq \mathit{cutp}(\mu_2)$ since every class from $P_u^1$ is included in a class from $P_u^2$: hence a variable from $Q'$ that is in the same class as a frontier variable or a constant in $P_u^1$ also is in $P_u^2$. It remains to prove that $\mathit{cutp}(\mu_2) \subseteq \mathit{cutp}(\mu_1)$. Let $x$ be a cutpoint of $\mu_2$ and $P_u^2(x)$ be the class of $x$ in $P_u^2$. Since $x$ is a cutpoint of $\mu_2$, there is a term $t$ in $P_u^2(x)$ that is a constant or a frontier variable. Since $P_u^1 \geq P_u^2$, we know that $P_u^1(x) \subseteq P_u^2(x)$. Let $t'$ be a term of $H'$ from $P_u^1(x)$ (there is at least one term of $H'$ and one term of $Q'$ in each class since the partition is part of a unifier of $H'$ and $Q'$). We are sure that $t'$ is not an existential variable because $t' \in P_u^2(x)$ and an existential variable cannot be in the same class as $t$ (Condition 2 in the definition of a piece-unifier), so $t'$ is a frontier variable or a constant, hence $x$ is a cutpoint of $\mu_1$. □

**Property 14** *Let $\mu_1 = (Q', H', P_u^1)$ and $\mu_2 = (Q', H', P_u^2)$ be two piece-unifiers such that $\mu_1 \geq \mu_2$. Then $\beta(Q, R, \mu_1) \geq \beta(Q, R, \mu_2)$.*

*Proof:* Let $u_1$ (resp. $u_2$) be a substitution associated with $P_u^1$ (resp. $P_u^2$). Since $P_u^1 \geq P_u^2$, there is a substitution $s$ such that $u_2 = s \circ u_1$ . Then $\beta(Q, R, \mu_2) = u_2(\mathit{body}(R)) \cup u_2(Q \setminus Q') = (s \circ u_1)(\mathit{body}(R)) \cup (s \circ u_1)(Q \setminus Q') = (s \circ u_1)(\mathit{body}(R) \cup (Q \setminus Q')) = s(u_1(\mathit{body}(R) \cup (Q \setminus Q'))) = s(\beta(Q, R, \mu_1))$. $s$ is thus a homomorphism from $\beta(Q, R, \mu_1)$ to $\beta(Q, R, \mu_2)$, hence $\beta(Q, R, \mu_1) \geq \beta(Q, R, \mu_2)$. □

The following lemma expresses that piece-based rewriting operator is prunable.

**Lemma 15** *If $Q_1 \geq Q_2$ then for any piece-unifier $\mu_2$ of $Q_2$ with $R$: either (i) $Q_1 \geq \beta(Q_2, R, \mu_2)$ or (ii) there is a piece-unifier $\mu_1$ of $Q_1$ with $R$ such that $\beta(Q_1, R, \mu_1) \geq \beta(Q_2, R, \mu_2)$.*

*Proof:* Let $h$ be a homomorphism from $Q_1$ to $Q_2$. Let $\mu_2 = (Q'_2, H'_2, P^2_u)$ be a piece-unifier of $Q_2$ with $R$, and let $u_2$ be a substitution associated with $P^2_u$. We consider two cases:

(i) If $h(Q_1) \subseteq (Q_2 \setminus Q'_2)$, then $u_2 \circ h$ is a homomorphism from $Q_1$ to $u_2(Q_2 \setminus Q'_2) \subseteq \beta(Q_2, R, \mu_2)$. Thus $Q_1 \geq \beta(Q_2, R, \mu_2)$.

(ii) Otherwise, let $Q'_1$ be the non-empty subset of $Q_1$ mapped by $h$ to $Q'_2$, i.e., $h(Q'_1) \subseteq Q'_2$, and $H'_1$ be the subset of $H'_2$ matched by $u_2$ with $u_2(h(Q'_1))$, i.e., $u_2(H'_1) = u_2(h(Q'1))$. Let $P^1_u$ be the partition on $\mathit{terms}(H'_1) \cup \mathit{terms}(Q'_1)$ such that two terms are in the same class of $P^1_u$ if these terms or their images by $h$ are in the same class of $P^2_u$ (i.e., for a term $t$, we consider $t$ if $t$ is in $Q'_1$, and $h(t)$ otherwise). By construction, $(Q'_1, H'_1, P^1_u)$ is a piece-unifier of $Q_1$ with $R$. Indeed, $P^1_u$ fulfills all the conditions of the piece-unifier definition since $P^2_u$ fulfills these conditions.

Let $u_1$ be a substitution associated with $P^1_u$. For each class $P$ of $P^1_u$ (resp. $P^2_u$), we call *selected element* the unique element $t$ of $P$ such that $u_1(t) = t$ (resp. $u_2(t) = t$). We build a substitution $s$, from the selected elements of the classes in $P^1_u$ which are variables, to the selected elements of the classes in $P^2_u$, as follows: for any class $P$ of $P^1_u$, let $t$ be the selected element of $P$: if $t$ is a variable of $H'_1$ then $s(t) = u_2(t)$, otherwise $s(t) = u_2(h(t))$ ($t$ occurs in $Q'_1$). Note that, for any term $t$ in $P^1_u$, we have $s(u_1(t)) = u_2(h(t))$.

We build now a substitution $h'$ from $\mathit{vars}(\beta(Q_1, R, \mu_1))$ to $\mathit{terms}(\beta(Q_2, R, \mu_2))$, by considering three cases according to the part of $\beta(Q_1, R, \mu_1)$ in which the variable occurs, i.e., in $Q_1$ but not in $Q'_1$, in $\mathit{body}(R)$ but not in $H'_1$, or in the remaining part corresponding to the images of $sep(Q'_1)$ by $u_1$:

1. if $x \in \mathit{vars}(Q_1) \setminus \mathit{vars}(Q'_1)$, $h'(x) = h(x)$;
2. if $x \in \mathit{vars}(\mathit{body}(R)) \setminus \mathit{vars}(H'_1)$, $h'(x) = u_2(x)$;
3. if $x \in u_1(\mathit{sep}(Q'_1))$(or alternatively $x \in u_1(\mathit{fr}(R) \cap \mathit{vars}(H'_1))$), $h'(x) = s(x)$ ;

We conclude by showing that $h'$ is a homomorphism from $\beta(Q_1, R, \mu_1) = u_1(\mathit{body}(R)) \cup u_1(Q_1 \setminus Q'_1)$ to $\beta(Q_2, R, \mu_2) = u_2(\mathit{body}(R)) \cup u_2(Q_2 \setminus Q'_2)$ with two points:

1. $h'(u_1(\mathit{body}(R))) = u_2(\mathit{body}(R))$. Indeed, for any variable $x$ of $\mathit{body}(R)$:
   - either $x \in \mathit{vars}(\mathit{body}(R)) \setminus \mathit{vars}(H'_1)$, hence $h'(u_1(x)) = h'(x) = u_2(x)$ ($u_1$ is a substitution from variables of $Q'_1 \cup H'_1$),
   - or $x \in \mathit{fr}(R) \cap \mathit{vars}(H'_1)$, hence $h'(u_1(x)) = s(u_1(x)) = u_2(h(x)) = u_2(x)$ ($h$ is a substitution from variables of $Q_1$).
2. $h'(u_1(Q_1 \setminus Q'_1)) \subseteq u_2(Q_2 \setminus Q'_2)$. We show that $h'(u_1(Q_1 \setminus Q'_1)) = u_2(h(Q_1 \setminus Q'_1)))$, and since $h(Q_1 \setminus Q'_1) \subseteq Q_2 \setminus Q'_2$, we have $h'(u_1(Q_1 \setminus Q'_1)) \subseteq u_2(Q_2 \setminus Q'_2)$. To show that $h'(u_1(Q_1 \setminus Q'_1)) = u_2(h(Q_1 \setminus Q'_1)))$, we point out that, for any variable $x$ from $Q_1 \setminus Q'_1$:
   - either $x \in \mathit{vars}(Q'_1)$, then $h'(u_1(x)) = s(u_1(x)) = u_2(h(x))$
   - or $x \in \mathit{vars}(Q_1) \setminus \mathit{vars}(Q'_1)$, then $h'(u_1(x)) = h'(x) = h(x) = u_2(h(x))$ ($u_1$ is a substitution from variables of $Q'_1 \cup H'_1$ and $u_2$ is a substitution from variables of $Q'_2 \cup H'_2$ and $h(x) \notin \mathit{vars}(Q'_2 \cup H'_2)$).

□

Given a query $Q$ and a set of rules $\mathcal{R}$, the *piece-based rewriting operator* computes the set of one-step piece-rewritings of $Q$ according to all piece-unifiers of $Q$ with a rule $R \in \mathcal{R}$. We are now able to show that this operator fulfills the desired properties introduced in Section 4.

**Theorem 16** *Piece-based rewriting operator is sound, complete and prunable; this property is still true if only most general piece-unifiers are considered.*

*Proof:* Soundness and completeness follow from Theorem 12. Prunability follows from Lemma 15. Thanks to Property 14, the proof remains true if most general piece-unifiers are considered. □

# 6 Exploiting Single-Piece Unifiers

We are now interested in the efficient computation of piece-based rewritings. We identify several sources of combinatorial explosion in the computation of the piece-unifiers between a query and a rule:

1. The problem of deciding whether there is a piece-unifier of a given query $Q$ with a given rule $R$ is NP-complete in the general case. NP-hardness is easily obtained by considering the case of a rule with an empty frontier: then, there is a piece-unifier between $Q$ and $R$ if and only if there is a homomorphism from $Q$ to $H = \mathit{head}(R)$, which is an NP-complete problem, $Q$ and $H$ being any sets of atoms.

2. The number of most general piece-unifiers can be exponential in $|Q|$, even if the rule head $H$ is restricted to a single atom. For instance, assume that each atom of $Q$ unifies with $H$ and forms its own piece; then there may be $2^{|Q|}$ piece-unifiers obtained by considering all subsets of $Q$.

3. The same atom in $Q$ may belong to distinct pieces according to distinct unifiers, as illustrated by the next example.

**Example 7** *Let $Q = r(u, v) \wedge q(v)$ and $R = p(x) \rightarrow r(x, y) \wedge r(y, x) \wedge q(y)$. Atom $r(u, v)$ belongs to two single-piece unifiers: $(\{r(u, v), q(v)\}, \{r(x, y), q(y)\}, \{\{u, x\}, \{v, y\}\})$ and $(\{r(u, v)\}, \{r(y, x)\}, \{\{u, y\}, \{v, x\}\})$. For an additional example, see Example 6, where $p(u, v)$ and $p(v, w)$ both belong to $\mu_1$ and $\mu_2$.*

To cope with this complexity, an idea is to rely on *single-piece* unifiers, i.e., piece-unifiers of the form $(Q', -, -)$ where $Q'$ is a single piece of $Q$. This section is devoted to the properties of rewriting operators exploiting this notion. We show that the rewriting operator based on single-piece (most general) unifiers is sound and complete. However, perhaps surprisingly, it is not prunable, which prevents to use it in the generic algorithm. To recover prunability, we will define the aggregation of single-piece unifiers, which provides us with a new rewriting operator, which has all the desired properties and generates rewriting sets with fewer components than the standard piece-unifier. Note, however, that this will not completely remove the second complexity source (i.e., the exponential number of unifiers to consider) since the number of aggregations of single-piece unifiers can still be exponential in the size of $Q$, even with atomic-head rules.

## 6.1 Single-Piece Based Operator

As expressed by the following theorem, (most general) single-piece unifiers provide a sound and complete operator.

**Theorem 17** *Given a BCQ $Q$ and a set of rules $\mathcal{R}$, the set of rewritings of $Q$ obtained by considering exclusively most general single-piece unifiers is sound and complete.*

*Proof:* See Appendix. □

The proof of this theorem is given in Appendix since it is not reused hereafter. Indeed, the restriction to single-piece unifiers is not compatible with selecting most general rewritings at each step, as performed in Algorithm 1. We present below some examples that illustrate this incompatibility.

**Example 8 (Basic example)** *Let $Q = p(y,z) \wedge p(z,y)$ and $R = r(x,x) \rightarrow p(x,x)$. There are two single-piece unifiers of $Q$ with $R$, $\mu_1 = (\{p(y,z)\}, \{p(x,x)\}, \{\{x,y,z\}\})$ and $\mu_2 = (\{p(z,y)\}, \{p(x,x)\}, \{\{x,y,z\}\})$, which yield the same rewriting, e.g. $Q_1 = r(x,x) \wedge p(x,x)$. There is also a two-piece unifier $\mu = (Q, \{p(x,x)\}, \{\{x,y,z\}\})$, which yields e.g. $Q' = r(x,x)$. A query equivalent to $Q'$ can be obtained from $Q_1$ by a further single-piece unification. Now, assume that we restrict unifiers to single-piece unifiers* and *keep most general rewritings at each step. Since $Q \geq Q_1$, $Q_1$ is not kept, hence $Q'$ will never be generated, whereas it is incomparable with $Q$.*

Concerning the preceding example, given $u_1$ and $u_2$ the substitutions respectively associated with $\mu_1$ and $\mu_2$, one may argue that $u_1(Q)$ is redundant and the same holds for $u_2(Q)$; hence, the problem would be solved by computing $u_1(Q) \backslash u_1(Q')$ instead of $u_1(Q \setminus Q')$ and making $u_1(Q)$ non-redundant (i.e., equal to $p(x,x)$) before computing $u_1(Q) \setminus u_1(Q')$, which would then be empty. However, the problem goes deeper, as illustrated by the next two examples.

**Example 9 (Ternary predicates)** *Let $Q = r(u,v,w) \wedge r(w,t,u)$ and $R = p(x,y) \rightarrow r(x,y,x)$. Again, there are two single-piece unifiers of $Q$ with $R$: $\mu_1 = (\{r(u,v,w)\}, \{r(x,y,x)\}, \{\{u,w,x\}, \{v,y\}\})$ and $\mu_2 = (\{r(w,t,u)\}, \{r(x,y,x)\}, \{\{u,w,x\}, \{t,y\}\})$. One obtains two rewritings more specific than $Q$, e.g., $Q_1 = p(x,y) \wedge r(x,v,x)$, and $Q_2 = p(x,y) \wedge r(x,t,x)$, which are isomorphic. There is also a two-piece unifier $(Q, \{r(x,y,x)\}, \{\{u,w,x\}, \{v,t,y\}\})$, which yields $p(x,y)$. If we remove $Q_1$ and $Q_2$, no query equivalent to $p(x,y)$ can be generated.*

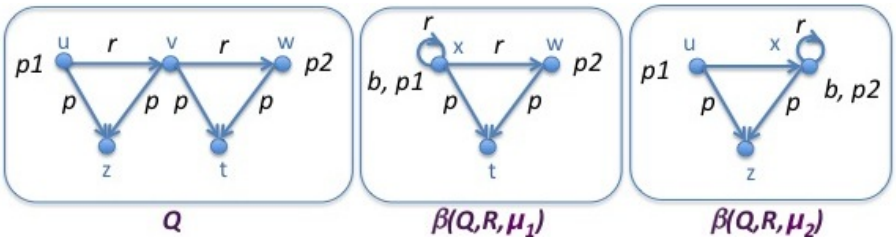


Figure 4: The queries in Example 10

**Example 10 (Very simple rule)** *This example has two interesting characteristics: (1) it uses unary/binary predicates only (2) it uses a very simple rule expressible with any lightweight description logic, i.e., a linear existential rule where no variable appears twice in the head or the body. Let* $Q = r(u,v) \wedge r(v,w) \wedge p(u,z) \wedge p(v,z) \wedge p(v,t) \wedge p(w,t) \wedge p_1(u) \wedge p_2(w)$ *(see Figure 4) and* $R = b(x) \rightarrow p(x,y)$*. Note that* $Q$ *is not redundant. There are two single-piece unifiers of* $Q$ *with* $R$*, say* $\mu_1$ *and* $\mu_2$*, with pieces* $Q'_1 = \{p(u,z), p(v,z)\}$ *and* $Q'_2 = \{p(v,t), p(w,t)\}$ *respectively. The obtained queries are pictured in Figure 4. These queries are both more specific than* $Q$*. The removal would prevent the generation of a query equivalent to* $r(x,x) \wedge p_1(x) \wedge p_2(x) \wedge b(x)$*, which could be generated from* $Q$ *with a two-piece unifier.*

**Property 18** *The single-piece-based operator is not prunable.*

*Proof:* Follows from the above examples. □

By Theorem 5 and Property 24, one can show that the conclusion of Lemma 3 (Section 4.2) is valid for single-piece unifiers, even though they are not prunable. This justifies that Lemma 3 is not enough to prove the correctness of Algorithm 1.

Nevertheless, single-piece unifiers can still be used as an algorithmic brick to compute more complex piece-unifiers, as shown in the next subsection.

## 6.2 Aggregated-Piece Based Operator

We first explain the ideas that underline aggregated single-piece unifiers. Let us consider the set of single-piece unifiers naturally associated with a piece-unifier $\mu$. If we successively apply each of these underlying single-piece unifiers, we may obtain a CQ strictly more general than $\beta(Q, R, \mu)$, as illustrated by the next example.

**Example 11** *Let* $R = p(x,y) \rightarrow q(x,y)$ *and* $Q = q(u,v) \wedge r(v,w) \wedge q(t,w)$*. Let* $\mu = (Q', H', P_u)$ *be a piece-unifier of* $Q$ *with* $R$ *with* $Q' = \{q(u,v), q(t,w)\}$*,* $H' = \{q(x,y)\}$ *and* $P_u = \{\{u,t,x\}, \{v,w,y\}\}$*.* $\beta(Q, R, \mu) = p(x,y) \wedge r(y,y)$*.* $Q'$ *has two pieces w.r.t.* $\mu$*:* $P_1 = \{q(u,v)\}$ *and* $P_2 = \{q(t,w)\}$*. If we successively compute the rewritings with the underlying single-piece unifiers* $\mu_{P_1}$ *and* $\mu_{P_2}$*, we obtain* $Q_s = \beta(\beta(Q, R, \mu_{P_1}), R, \mu_{P_2}) = \beta(p(x,y) \wedge r(y,w) \wedge q(t,w), R, \mu_{P_2}) = p(x,y) \wedge r(y,y') \wedge p(x',y')$*, which is strictly more general than* $\beta(Q, R, \mu)$*.*

Given a set $\mathcal{U}$ of "compatible" single-piece unifiers of a query $Q$ with a rule (the notion of "compatible" will be formally defined below), we can thus distinguish between the usual piece-unifier performed on the union of the pieces from the unifiers in $\mathcal{U}$ and an "aggregated unifier" that would correspond to a sequence of applications of the unifiers in $\mathcal{U}$. This latter unifier is more interesting than the piece-unifier because, as illustrated by Example 11, it avoids generating some rewritings which are too specific. We will thus rely on the aggregation of single-piece unifiers to recover prunability.

Note that, in this paper, we combine single-piece unifiers of the *same* rule whereas in [KLMT13] we consider the possibility of combining unifiers of distinct rules (and thus compute rewritings from distinct rules in a single step). We keep below the definitions introduced in [KLMT13], while pointing out that, in the context of this paper, the rules $R_1 \ldots R_k$ in the definitions are necessarily copies of the same rule $R$. Intuitively, an aggregated unifier of $R$ is a piece-unifier of a new rule built by aggregating copies of $R$ (as formally expressed by next Property 19).

**Definition 14 (Aggregation of a set of rules)** *Let $\mathcal{R} = \{R_1 \dots R_k\}$ be a set of rules, with pairwise disjoint sets of variables. The* aggregation *of $\mathcal{R}$, denoted by $R_1 \diamond \dots \diamond R_k$, is the rule $\mathit{body}(R_1) \wedge \dots \wedge \mathit{body}(R_k) \rightarrow \mathit{head}(R_1) \wedge \dots \wedge \mathit{head}(R_k)$.*

**Definition 15 (Compatible set of piece-unifiers, Aggregated unifier)**
*Let $\mathcal{U} = \{\mu_1 = (Q'_1, H'_1, P_1) \dots \mu_k = (Q'_k, H'_k, P_k)\}$ be a set of piece-unifiers of $Q$ with rules $R_1 \dots R_k$ respectively, where the rules have pairwise disjoint sets of variables (in particular, for all $1 \leq i, j \leq k, i \neq j$, it holds that $\mathit{vars}(H'_i) \cap \mathit{vars}(H'_j) = \emptyset$).*

- *Set $\mathcal{U}$ is said to be* compatible *if (1) all $Q'_i$ and $Q'_j$ are pairwise disjoint; (2) the join of $P_1 \dots P_k$ is admissible.*
- *Given such a compatible set $\mathcal{U}$, an* aggregated unifier *of $Q$ with $R_1 \dots R_k$ w.r.t. $\mathcal{U}$ is $\mu = (Q', H', P)$ where: (1) $Q' = Q'_1 \cup \dots \cup Q'_k$; (2) $H' = H'_1 \cup \dots \cup H'_k$; (3) $P$ is the join of $P_1 \dots P_k$. It is said to be* single-piece *if all the piece-unifiers in $\mathcal{U}$ are single-piece. It is said to be* most general *if all the piece-unifiers in $\mathcal{U}$ are most general.*

**Property 19** *Let $Q$ be a BCQ and $\mathcal{U} = \{\mu_1 = (Q'_1, H'_1, P_1) \dots \mu_k = (Q'_k, H'_k, P_k)\}$ be a compatible set of piece-unifiers of $Q$ with $R_1 \dots R_k$. Then, the aggregated unifier of $\mathcal{U}$ is a piece-unifier of $Q$ with the aggregation of $\{R_1 \dots R_k\}$.*

*Proof:* We show that the aggregated unifier $\mu = (Q', H', P_u)$ of $\mathcal{U}$ satisfies the conditions of the definition of a piece-unifier (Definition 11). Condition 1 is fulfilled since, by definition of compatibility, the join of $P_1 \dots P_k$ is admissible. Condition 2 is satisfied as well, because, since $P_1 \dots P_k$ satisfy it, so does their join. Indeed, if a class contains an existential variable, it cannot be merged with another by aggregation because its other terms are non-separating variables, hence do not appear in other classes. Concerning the last condition, for all $1 \leq i \leq k$, we have $u_i(H'_i) = u_i(Q'_i)$, where $u_i$ is a substitution associated with $P_i$. Since $Q' = \bigcup_{i=1}^{k} Q'_i$ and $H' = \bigcup_{i=1}^{k} H'_i$ we know that, for any substitution $u$ associated with $P_u$, we have $u(H') = u(Q')$. □

According to this property, the *rewriting* associated with an aggregated unifier $\mu$ can be defined as $\beta(Q, R_1 \diamond \dots \diamond R_k, \mu)$. It corresponds to the rewriting obtained by applying the piece-unifiers associated with the $R_i$ one after the other, as illustrated by the next example.

**Example 12** *Consider again Example 11. Let $R' = p(x', y') \rightarrow q(x', y')$ be a copy of $R$. The aggregation $R \diamond R'$ is the rule $p(x, y) \wedge p(x', y') \rightarrow q(x, y) \wedge q(x', y')$. Let $\mathcal{U} = \{\mu_{P_1}, \mu_{P_2}\}$ where $\mu_{P_1} = (\{q(u, v)\}, \{q(x, y)\}, \{\{u, x\}, \{v, y\}\})$ and $\mu_{P_2} = (\{q(t, w)\}, \{q(x', y')\}, \{\{t, x'\}, \{w, y'\}\})$ . The aggregated unifier of $Q$ with $R, R'$ w.r.t. $\mathcal{U}$ is $(\{q(u, v), q(t, w)\}, \{q(x, y), q(x', y')\}, \{\{u, x\}, \{v, y\}, \{t, x'\}, \{w, y'\}\})$. The associated rewriting of $Q$ is $p(x, y) \wedge r(y, y') \wedge p(x', y')$, which is equal to the rewriting $Q_s$ in Example 11.*

The difference between a piece-unifier and an aggregated unifier of $Q$ with $R$ can also be explained as follows: to build a piece-unifier of $Q$ with $R$, we consider partitions of $\mathit{terms}(Q) \cup \mathit{terms}(\mathit{head}(R))$, while in the aggregation operation we consider partitions of $\mathit{terms}(Q) \cup \bigcup_{i=1}^{k} \mathit{terms}(\mathit{head}(R_i))$, where $k$ is the number of considered single-piece unifiers, and each $R_i$ is safely renamed from $R$. In other words, if, in the definition of an aggregated unifier, we assumed that the $R_1 \dots R_k$ had been exactly $R$, instead of safely renamed copies of $R$, then the aggregation of $R_1 \dots R_k$ would have

been exactly $R$ after removal of duplicate atoms, and the aggregated unifier would have been the usual piece-unifier.

The next property shows that, from any piece-unifier $\mu$, one can build a most general single-piece aggregated unifier, which produces a rewriting more general than the one produced by $\mu$.

**Property 20** *For any piece-unifier $\mu$ of $Q$ with $R$, there is a most general single-piece aggregated unifier $\mu_\diamond$ of $Q$ with $R_1 \dots R_k$ copies of $R$ such that $\beta(Q, R_1 \diamond \dots \diamond R_k, \mu_\diamond) \geq \beta(Q, R, \mu)$.*

*Proof:* Let $Q'_1, \dots, Q'_k$ be the pieces of $Q'$ according to $\mu = (Q', H', P_u)$ and let $u$ be a substitution associated with $P_u$. Let $R_1 \dots R_k$ be safely renamed copies of $R$. Let $h_i$ denote the variable renaming used to produce $R_i$ from $R$. Let $\mathcal{U} = \{\mu_1 = (Q'_1, H'_1, P^1_u), \dots, \mu_k = (Q'_k, H'_k, P^k_u)\}$ be a set of piece-unifiers of $Q$ with $R_1, \dots, R_k$ built as follows for all $i$:

- $H'_i$ is the image by $h_i$ of the subset of $H'$ unified by $u$ with $Q'_i$
- let $h_i(P_u)$ be the partition built from $P_u$ by replacing each $x \in \mathit{vars}(H')$ by $h_i(x)$; then, $P^i_u$ is obtained from $h_i(P_u)$ by (1) restricting it to the terms of $Q'_i$ and $H'_i$, and (2) refining it as much as possible while keeping the property that $u_i(H'_i) = u_i(Q'_i)$, where $u_i$ is a substitution associated with the partition.

For any $\mu_i = (Q'_i, H'_i, P^i_u)$ we immediately check that:

1. $\mu_i$ is a most general piece-unifier.
2. $\mu_i$ is a single-piece unifier.
3. for all $\mu_j \in \mathcal{U}$, with $\mu_i \neq \mu_j$, $\mu_j$ and $\mu_i$ are compatible.

Let $\mu_\diamond = (Q'_\diamond, H'_\diamond, P^\diamond_u)$ be the aggregated unifier of $Q$ with $R_1, \dots, R_k$ w.r.t. $\mathcal{U}$. Note that $Q'_\diamond = Q'$. The above properties fulfilled by any $\mu_i$ from $\mathcal{U}$ ensure that $\mu_\diamond$ is a most general single-piece aggregated unifier.

We note $R_\diamond = R_1 \diamond \dots \diamond R_k$. It remains to prove that $\beta(Q, R_\diamond, \mu_\diamond) \geq \beta(Q, R, \mu)$. Let $u_\diamond$ be a substitution associated with $P^\diamond_u$. For each class $P$ of $P_u$ (resp. $P^\diamond_u$), we call *selected element* the unique element $t$ of $P$ such that $u(t) = t$ (resp. $u_\diamond(t) = t$).

We build a substitution $s$, from the selected elements in $P^\diamond_u$ which are variables, to the selected elements in $P_u$, as follows: for any class $P$ of $P^\diamond_u$, let $t$ be the selected element of $P$: if $t$ is a variable of $Q'$, then $s(t) = u(t)$; else $t$ is a variable of a $H'_i$: then $s(t) = u(h_i^{-1}(t))$. Note that for any term $t$ in $P^\diamond_u$, there is a variable renaming $h_i$ such that $s(u_\diamond(t)) = u(h_i^{-1}(t))$ (if $t$ is a constant or a variable from $\mathit{vars}(Q)$ then any $h_i$ can be chosen).

We build now a substitution $h$ from $\mathit{vars}(\beta(Q, R_\diamond, \mu_\diamond))$ to $\mathit{terms}(\beta(Q, R, \mu))$, by considering three cases according to the part of $\beta(Q, R_\diamond, \mu_\diamond)$ in which the variable occurs, i.e., in $Q$ but not in $Q'$, in $\mathit{body}(R_i)$ but not in $H'_i$, or in the remaining part corresponding to the images of $sep(Q')$ by $u_\diamond$:

1. if $x \in \mathit{vars}(Q) \setminus \mathit{vars}(Q')$, $h(x) = x$;
2. if $x \in \mathit{vars}(\mathit{body}(R_i)) \setminus \mathit{vars}(H'_i)$, $h(x) = h_i^{-1}(x)$;
3. if $x \in u_\diamond(\mathit{sep}(Q'))$(or alternatively $x \in u_\diamond(\mathit{fr}(R_\diamond) \cap \mathit{vars}(H'_\diamond))$), $h(x) = s(x)$ ;

We conclude by showing that $h$ is a homomorphism from $\beta(Q, R_\diamond, \mu_\diamond) = u_\diamond(\mathit{body}(R_1) \cup \ldots \cup \mathit{body}(R_k)) \cup u_\diamond(Q \setminus Q')$ to $\beta(Q, R, \mu) = u(\mathit{body}(R)) \cup u(Q \setminus Q')$, with two points:

1. for all $i$, $h(u_\diamond(\mathit{body}(R_i))) = u(\mathit{body}(R))$. Indeed, for any variable $x \in \mathit{vars}(\mathit{body}(R_i))$:
   - either $x \in \mathit{vars}(\mathit{body}(R_i)) \setminus \mathit{vars}(H'_i)$, hence $h(u_\diamond(x)) = h(x) = h_i^{-1}(x) = u(h_i^{-1}(x))$ ($u$ does not substitute the variables from $\mathit{vars}(\mathit{body}(R)) \setminus \mathit{vars}(H')$),
   - or $x \in \mathit{fr}(R_i) \cap \mathit{vars}(H'_i)$, hence $h(u_\diamond(x)) = s(u_\diamond(x)) = u(h_i^{-1}(x))$;
2. $h(u_\diamond(Q \setminus Q')) = u(Q \setminus Q')$. Indeed, for any variable $x \in \mathit{vars}(Q \setminus Q')$:
   - either $x \in \mathit{vars}(Q')$, then $h(u_\diamond(x)) = s(u_\diamond(x)) = u(h_i^{-1}(x)) = u(x)$ ($h_i^{-1}$ does not substitute the variables from $Q$),
   - or $x \in \mathit{vars}(Q) \setminus \mathit{vars}(Q')$, then $h(u_\diamond(x)) = h(x) = x = u(x)$ ($u_\diamond$ and $u$ do not substitute the variables from $\mathit{vars}(Q) \setminus \mathit{vars}(Q')$).

□

We call *single-piece aggregator* the rewriting operator that computes the set of one-step rewritings of a query $Q$ by considering all the most general single-piece aggregated unifiers of $Q$.

**Theorem 21** *The single-piece aggregator is sound, complete and prunable.*

*Proof:* Soundness comes from Property 19 and from the fact that, for any set of rules $\mathcal{R}$, let $R$ be the aggregation of $\mathcal{R}$, one has $\mathcal{R} \models R$. Completeness and prunability rely on the fact that the piece-based rewriting operator fulfills these properties, and the fact that for any queries $Q$ and $Q'$ and any rule $R$, if $Q' = \beta(Q, R, \mu)$, where $\mu$ is a piece-unifier, then the query $Q''$ obtained with the single-piece aggregator corresponding to $\mu$ is more general than $Q'$, as expressed by Property 20. □

# 7 Detailed Algorithms and Experiments

In this section, we first detail on the computation of all the most general single-piece unifiers of a query $Q$ with a rule $R$, and explain how we use them to compute all the single-piece aggregators. Then, we focus on the specific case of unification with atomic-head rules, for which the computation is simpler. Last, we report first experiments.

## 7.1 Computing single-piece unifiers and their aggregation

We first introduce the notion of a *pre-unifier*, which is weaker than a piece-unifier. To become a piece-unifier, a pre-unifier has to satisfy an additional constraint on the separating variables of the unified subset of $Q$.

**Definition 16 (Valid Partition)** *Let $Q$ be a BCQ, $R$ be a rule, $Q' \subseteq Q$, $H' \subseteq \mathit{head}(R)$, and $P_u$ be a partition on $\mathit{terms}(Q') \cup \mathit{terms}(H')$. $P_u$ is* valid *if no class of $P_u$ contains two constants, or two existential variables of $R$, or a constant and an existential variable of $R$, or an existential variable of $R$ and a frontier variable of $R$.*

**Definition 17 (Pre-unifier)** *Let $Q$ be a BCQ, $R$ be a rule, $Q' \subseteq Q$, $H' \subseteq$ head$(R)$, and $P_u$ be a partition on* terms$(Q') \cup$ terms$(H')$. *Then $\mu = (Q', H', P_u)$ is a* pre-unifier *of $Q$ with $R$ if* (1) *$P_u$ is valid, and* (2) *given a substitution $u$ associated with $P_u$, it holds that $u(H') = u(Q')$.*

The next definition introduces the notion of *sticky* variables, which are the variables of $Q'$ that prevent $Q'$ to be a piece.

**Definition 18 (Sticky Variables)** *Let $Q$ be a BCQ, $R$ be a rule, $Q' \subseteq Q$, $H' \subseteq$ head$(R)$ and $P_u$ be a partition on* terms$(Q') \cup$ terms$(H')$. *The* sticky variables *of $Q'$ in $P_u$ w.r.t. $Q$ and $R$, denoted by $sticky(Q', P_u)$, are the separating variables of $Q'$ that occur in a class of $P_u$ containing an existential variable of $R$.*

The next property ensures that a pre-unifier without sticky variables is a piece-unifier, and reciprocally. Its proof follows from the definitions.

**Property 22** *Let $Q$ be a BCQ, $R$ be a rule, $Q' \subseteq Q$, $H' \subseteq$ head$(R)$, and $P_u$ be a partition on* terms$(Q') \cup$ terms$(H')$. *Then $\mu = (Q', H', P_u)$ is a piece-unifier of $Q$ with $R$ iff $\mu$ is a pre-unifier and $sticky(Q', P_u) = \emptyset$.*

The fact that we can first build pre-unifiers, then check the absence of sticky variables, suggests an incremental method to compute all the most general single piece-unifiers of $Q$ with $R$.

The first step consists in computing all the most general pre-unifiers of an atom $a \in Q$ with an atom $b \in$ head$(R)$ with the same predicate. The partition on the terms of these atoms associated with their unification has to be valid. The next definition defines formally this notion of partition.

**Definition 19 (Partition by Position)** *Let $A$ be a set of atoms with the same predicate $p$. The* partition by position *associated with $A$, denoted by $P_p(A)$, is the partition on* terms$(A)$ *such that two terms of $A$ occurring in the same position $i$ ($1 \leq i \leq arity(p)$) are in the same class of $P_p(A)$.*

Hence, the partition by position associated with $\{a, b\}$ has to be valid. We denote by $APU$ the set of all the most general atomic pre-unifiers, i.e., $APU = \{\mu = (\{a\}, \{b\}, P) \mid a \in Q, b \in$ head$(R)$, *and $\mu$ is a pre-unifier of $Q$ with $R$* $\}$. Algorithm 2 details the computation of $APU$.

We then use $APU$ to compute the set of all the most general single-piece unifiers of $Q$ with $R$, denoted by $SPU$. Each atomic pre-unifier of $APU$ is incrementally extended in all possible ways with other atomic pre-unifiers of $APU$, which contain "missing" atoms of $Q$ with respect to sticky variables. Extending pre-unifier $(Q_1, H_1, P_1)$ with pre-unifier $(Q_2, H_2, P_2)$ consists in merging both pre-unifiers to obtain a new pre-unifier $(Q_1 \cup Q_2, H_1 \cup H_2, join(P_1, P_2))$; this extension can be performed if and only if the join of $P_1$ and $P_2$ is a valid partition; if the obtained pre-unifier has no sticky variable, it is a single piece-unifier.

Next algorithms 3, 4 and 5 detail the computation of $SPU$. Algorithm 3 is the main algorithm. It first uses Algorithm 2 to compute $APU$, then, for each atomic pre-unifier $\mu \in APU$, it calls Algorithm 4, which computes the single-piece unifiers extending $\mu$. Algorithm 4 first checks if $\mu$ contains sticky variables: if it it is the case, this single-piece unifier is returned, otherwise the algorithm is recursively called, after a call to Algorithm 5 to obtain a set of candidate extensions of $\mu$.

**Algorithm 2:** COMPUTATION OF $APU$, THE SET OF THE MOST GENERAL ATOMIC PRE-UNIFIERS

**Data**: A BCQ $Q$ and a rule $R$
**Result**: The set of the most general pre-unifiers of an atom of $Q$ with an atom of *head*$(R)$

```
begin
    APU ← ∅;
    foreach a ∈ Q do
        foreach b ∈ head(R) do
            if predicate(a) = predicate(b) and P_p({a, b}) is valid then
                APU ← APU ∪ {({a}, {b}, P_p({a, b}))}
    return APU
```

**Algorithm 3:** COMPUTATION OF $SPU$, THE SET OF THE MOST GENERAL SINGLE-PIECE UNIFIERS

**Data**: A BCQ $Q$ and a rule $R$
**Result**: The set of most general single-piece unifiers of $Q$ with $R$

```
begin
    SPU ← ∅;
    APU ← computeAPU(Q, R) ; // see Algorithm 2
    while APU ≠ ∅ do
        choose and remove ({a}, {b}, P) from APU;
        SPU ← SPU ∪ SPUext({a}, {b}, P); // see Algorithm 4
    return SPU;
```

Finally, the set of all the single-piece aggregators of $Q$ with $R$ is obtained by aggregating the unifiers from all non-empty compatible subsets of $SPU$. For optimisation reasons, this set is incrementally computed as follows:

1. Let $U_1 = SPU = \{\mu_1, \ldots, \mu_k\}$; the $\mu_i$ are called 1-unifiers.
2. For $i = 2$ to the greatest possible rank (i.e., as long as $U_i$ is not empty): let $U_i$ be the set of all $i$-unifiers obtained by aggregating an $(i-1)$-unifier from $U_{i-1}$ and a single-piece unifier from $U_1$.
3. Return the union of all the $U_i$ obtained.

## 7.2 The Specific Case of Atomic-Head Rules

Rules with an *atomic head* are often considered in the literature, specifically in logic programming or in deductive databases. One may ask if piece-unification become simpler in this specific case. In fact, considering atomic-head rules does not simplify the definition of a piece-unifier in itself, but its computation. Indeed, there is now a unique way of associating any atom from $Q$ with the head of a rule. It follows that deciding whether there is a piece-unifier of $Q$ with a rule can be done in linear time with respect to the size of $Q$ (whereas it is NP-complete in the general case) and each

**Algorithm 4:** COMPUTATION OF THE MOST GENERAL SINGLE-PIECE UNIFIERS EXTENDING A GIVEN PRE-UNIFIER

**Access**: $Q$, $R$ and $APU$ declared in the calling algorithm 3
**Data**: $(Q', H', P')$ a pre-unifier of $Q$ with $R$
**Result**: The set of the most general single-piece unifiers extending $(Q', H', P')$
**begin**
  **if** $sticky(Q', P') = \emptyset$ **then**
    **return** $\{(Q', H', P')\}$ `// it is a single-piece unifier`
  **else**
    $Q_{add} \leftarrow \{a \in Q \setminus Q' \mid vars(a) \cap sticky(Q', P') \neq \emptyset\}$;
    $Ext \leftarrow$ Extend($(Q', H', P')$, $Q_{add}$, $APU$) ; `// see Algorithm 5`
    $EPU \leftarrow \emptyset$;
    **foreach** $(Q_{ext}, H_{ext}, P_{ext}) \in Ext$ **do**
      $EPU \leftarrow EPU \cup$ SPUext($Q_{ext}, H_{ext}, P_{ext}$);
      `// recursive call to Algorithm 4`
    **return** EPU ;

atom belongs to a single piece, thus the set of all single-piece unifiers of $Q$ with a rule can be computed in polynomial time.

More precisely, if a rule $R$ has an atomic head, then every atom in $Q$ participates in *at most one* most general single-piece unifier of $Q$ with $R$ (up to bijective variable renaming). This is is a corollary of the next property.

**Property 23** *Let $R$ be an atomic-head rule and $Q$ be a BCQ. For any atom $a \in Q$, there is at most one $Q' \subseteq Q$ such that $a \in Q'$ and $Q'$ is a piece for a piece-unifier of $Q$ with $R$.*

*Proof:* We prove by contradiction that two single-piece unifiers cannot share an atom of $Q$. Assume there are $Q'_1 \subseteq Q$ and $Q'_2 \subseteq Q$ such that $Q'_1 \neq Q'_2$ and $Q'_1 \cap Q'_2 \neq \emptyset$, and $\mu_1 = (Q'_1, H, P^1_u)$ and $\mu_2 = (Q'_2, H, P^2_u)$ two single-piece-unifiers of $Q$ with $R$, with $H = head(R)$. Since $Q'_1 \neq Q'_2$, one has $Q'_1 \setminus Q'_2 \neq \emptyset$ or $Q'_2 \setminus Q'_1 \neq \emptyset$. Assume $Q'_1 \setminus Q'_2 \neq \emptyset$. Let $A = Q'_1 \cap Q'_2$ and $B = Q'_1 \setminus A$. There is at least one variable $x \in vars(A) \cap vars(B)$ such that there is an existential variable $e$ of $head(R)$ in the class of $P^1_u$ containing $x$ (otherwise $\mu_1$ has more than one piece). Since $H$ is atomic, there is a unique way of associating any atom with $H$, thus the class of $P^2_u$ containing $x$ contains $e$ as well. It follows that $Q'_2$ is not a piece since an atom of $A$ and an atom of B share the variable $x$ unified with an existential variable in $\mu_2$, while $A$ is included in $Q'_2$ and $B$ is not. □

The fact that an atom from $Q$ participates in at most one most general single-piece unifier allows some algorithmic improvements. Indeed, when a piece-unifier of $Q'$ with $head(R)$ is successfully built, all the atoms of $Q'$ can be removed from the set of atoms to be considered in the computation of the next piece-unifiers. Furthermore, there is a unique way of associating any atom from $Q$ with $head(R)$, hence there is only one pre-unifier of $Q'$ with $head(R)$. Algorithm 6 exploits these specific aspects to compute all the single-piece unifiers of a query with an atomic-head rule.

**Example 13** *Let $R = q(x) \rightarrow p(x, y)$ and $Q = p(u, v) \wedge p(v, t)$. Let us start from $p(u, v)$: this atom is unifiable with head$(R)$ and $p(v, t)$ necessarily belongs to*

**Algorithm 5:** COMPUTATION OF THE PRE-UNIFIERS EXTENDING A GIVEN PRE-UNIFIER W.R.T. TO A GIVEN SET OF ATOMS

**Data**: $(Q', H', P')$ a pre-unifier of $Q$ with $R$,$Q_{add}$ a subset of $Q$ (disjoint from $Q'$), $APU$ a set of atomic pre-unifiers
**Result**: The set of pre-unifiers extending $(Q', H', P')$ w.r.t. $Q_{add}$ and $APU$
**begin**
  **if** $Q_{add} = \emptyset$ **then**
    **return** $\{(Q', H', P')\}$
  **else**
    $Ext \leftarrow \emptyset$;
    choose an atom $a \in Q_{add}$;
    **foreach** $(a, b, P_a) \in APU$ **do**
      **if** $join(P', P_a)$ *is valid* **then**
        $Ext \leftarrow Ext \cup$ Extend$((Q' \cup \{a\}, H' \cup \{b\}, join(P, P_a))$, $Q_{add} \setminus \{a\}$, $APU)$ ; `// recursive call to Algorithm 5`
    **return** $Ext$;

*the same piece-unifier (if any) because* $v \in sticky(\{p(u,v)\}, \{\{u,x\},\{v,y\}\})$*; indeed,* $v$ *is in the same class as the existential variable* $y$*; however,* $\{p(u,v), p(v,t)\}$ *is not unifiable with* head$(R)$ *because, since* $v$ *occurs at the first and at the second position of a* $p$ *atom,* $x$ *and* $y$ *should be unified, which is not possible, since* $y$ *is an existential variable; thus,* $p(u,v)$ *does not belong to any piece-unifier with* $R$*. However,* $p(v,t)$ *still has to be considered. Let us start from it:* $p(v,t)$ *is unifiable with* head$(R)$ *and forms its own piece because* sticky$(\{p(v,t)\}\{\{v,x\},\{t,y\}\})$ *is empty; indeed,* $t$ *is in the same class as the existential variable* $y$*, but does not occur in any other atom. Hence, there is a single (most general) piece-unifier of* $Q$ *with* $R$*, namely* $(\{p(v,t)\}, \{p(x,y)\}, \{\{v,x\},\{t,y\}\})$.

It should be noted that any existential rule can be decomposed into an equivalent set of rules with atomic head by introducing a new predicate, which gathers the variables of the original head (e.g. [CGK08, BLMS09]). Hence, the restriction to atomic-head rules can be made without loss of expressivity. Now, the question is whether it is more efficient to directly process rules with complex heads, or to decompose them into atomic-head rules and benefit from a simpler computation of piece-unifiers. The experiments reported below clearly show that the former choice is better.

## 7.3 Experiments and Perspectives

The query rewriting algorithm, instantiated with the rewriting operator described in the preceding section, has been implemented in Java. Since benchmarks dedicated to existential rules are not available yet, first experiments were carried out with sets of existential rules obtained by translation from ontologies expressed in the description logic DL-Lite$_{\mathcal{R}}$, namely ADOLENA (A), STOCKEXCHANGE (S), UNIVERSITY (U) and VICODI (V). This benchmark was introduced in [PUHM09] and then used in several papers, e.g., [GOP11, CTS11, ISG12, KLMT12]. Ontologies A and U contain some rules with multiple heads; the ontologies obtained by decomposing rules into

**Algorithm 6:** COMPUTATION OF ALL THE MOST GENERAL SINGLE-PIECE UNIFIERS IN THE CASE OF ATOMIC-HEAD RULES

**Data**: A BCQ $Q$ and an atomic-head rule $R$
**Result**: The set of most general single-piece unifiers of $Q$ with $R$
**begin**
  $U \leftarrow \emptyset$; `// resulting set`
  $A \leftarrow \{a \in Q \mid predicate(a) = predicate(\mathsf{head}(R))\}$;
  **while** $A \neq \emptyset$ **do**
    $chooseanatoma \in A$ ;
    $Q' \leftarrow \{a\}$ ;
    **while** $Q' \subseteq A$ *and there is a pre-unifier* $(Q', \mathsf{head}(R), P)$ *and* sticky$(Q', P) \neq \emptyset$ **do**
      $Q' \leftarrow Q' \cup \{a' \in Q \mid a'$ contains a variable from *sticky*$(Q', P)\}$ ;
    **if** $Q' \subseteq A$ *and there is a pre-unifier* $(Q', \mathsf{head}(R), P)$ **then**
      $U \leftarrow U \cup \{(Q', \mathsf{head}(R), P)\}$ ;
      $A \leftarrow A \setminus Q'$
    **else**
      $A \leftarrow A \setminus \{a\}$
  **return** $U$

atomic-head rules are respectively known as AX and UX. Additionally, we considered the translation of a larger ontology, the DL-Lite version of OpenGalen2[8] (G), which contains more than 50k rules. Each ontology is provided with five handcrafted queries.

In [KLMT12], we compared with other systems concerning the size of the output and pointed out that none of the existing systems output a complete set of rewritings. However, beside the fact that these systems have evolved since then, one can argue that the size of the rewriting set should not be a decisive criterion (indeed, assuming that the systems are sound and complete, a minimal rewriting set can be obtained by selecting most general elements, see Theorem 1). Therefore, other criteria have to be taken into account, such as the runtime or the total number of CQs generated during the rewriting process.

| | Time (ms) | | # Output | | # Generated | |
|---|---|---|---|---|---|---|
| | A | AX | A | AX | A | AX |
| Q1 | 170 | 330 | 27 | 41 | 459 | 720 |
| Q2 | 90 | 4900 | 50 | 1431 | 171 | 4567 |
| Q3 | 240 | 47290 | 104 | 4466 | 316 | 13838 |
| Q4 | 440 | 28620 | 224 | 3159 | 826 | 14526 |
| Q5 | 2100 | 1**h**36 | 624 | 32921 | 2416 | 215523 |
| | U | UX | U | UX | U | UX |
| Q1 | 0 | 10 | 2 | 5 | 1 | 4 |
| Q2 | 0 | 0 | 1 | 1 | 105 | 120 |
| Q3 | 10 | 20 | 4 | 12 | 42 | 155 |
| Q4 | 1370 | 4190 | 2 | 5 | 2142 | 4720 |
| Q5 | 20 | 20 | 10 | 25 | 153 | 351 |

Table 1: Impact of rule decomposition

[8] `http://www.opengalen.org/`

All tests reported here were performed on a DELL machine with a processor at 3.60 GHz and 16 GB of RAM, with 4 GB allocated to the Java Virtual Machine.

Table 1 reports the behavior of the rewriting algorithm on A vs AX and U vs UX with respect to three parameters: the runtime, the size of the output (number of CQs) and the number of generated CQs. The size of the output for AX and UX is before elimination of queries containing auxiliary predicates. The generated CQs are all the rewritings built during the rewriting process (excluding the initial query and possibly including some multi-occurrences of the same rewritings). We can see that avoiding rule decomposition makes a substantial difference. The gain is particularly striking with $Q_5$ on A / AX with respect to all three parameters (the runtime is 21 seconds for A and 1 hour and 36 minutes for AX, the size of the output is more than 52 times larger for AX before elimination of useless queries, and the number of generated queries is 89 times larger for AX). Moreover, we point out that only 29 / 102 rules in A and 5 / 77 rules in U have multiple heads, with only 2 atoms; we can reasonably expect that the gain increases with the proportion of multiple-head rules and the size of rule heads.

| Rules | Query | # Output | # Generated | # Explored | Time (ms) |
|---|---|---|---|---|---|
| A | Q1 | 27 | 459 | 74 | 170 |
| | Q2 | 50 | 171 | 70 | 90 |
| | Q3 | 104 | 316 | 104 | 240 |
| | Q4 | 224 | 826 | 256 | 440 |
| | Q5 | 624 | 2416 | 624 | 2100 |
| S | Q1 | 6 | 9 | 6 | 0 |
| | Q2 | 2 | 137 | 23 | 10 |
| | Q3 | 4 | 275 | 20 | 40 |
| | Q4 | 4 | 450 | 58 | 90 |
| | Q5 | 8 | 688 | 44 | 110 |
| U | Q1 | 2 | 1 | 2 | 0 |
| | Q2 | 1 | 105 | 32 | 0 |
| | Q3 | 4 | 42 | 10 | 10 |
| | Q4 | 2 | 2142 | 556 | 1370 |
| | Q5 | 10 | 153 | 14 | 20 |
| V | Q1 | 15 | 14 | 15 | 0 |
| | Q2 | 10 | 9 | 10 | 0 |
| | Q3 | 72 | 117 | 72 | 30 |
| | Q4 | 185 | 328 | 185 | 110 |
| | Q5 | 30 | 59 | 30 | 10 |
| G | Q1 | 2 | 2 | 2 | 10 |
| | Q2 | 1152 | 1275 | 1152 | 1090 |
| | Q3 | 488 | 1514 | 488 | 1050 |
| | Q4 | 147 | 154 | 147 | 30 |
| | Q5 | 324 | 908 | 324 | 1000 |

Table 2: Generated queries with the single-piece aggregator

Table 2 presents the size of the output, the number of generated CQs and the number of explored CQs for each ontology (as well as the runtime for information, see also Table 4). Note that, since subsumed rewritings are removed at each step of the breadth-first algorithm, only some of the rewritings generated at a given step are explored at the next step. We can see that the number of generated queries can be large with respect to the cardinality of the output, which is less marked for explored queries.

| Ontology | # Rules | # Hierarchical rules |
|---|---|---|
| A | 102 | 72 |
| S | 52 | 16 |
| U | 77 | 36 |
| V | 222 | 202 |
| G | 50764 | 26980 |

Table 3: Types of rules in the ontologies

Our query rewriting tool is able to process any kind of existential rules. There is of course a price to pay for this expressivity, in terms of complexity of the involved mechanisms and time efficiency. We consider the algorithms presented in this paper as basic versions, which can be further improved in various ways, for instance by processing some specific kinds of rules in a specific way. Let us illustrate this with the example of rules expressing taxonomies. Indeed, a large part of currently available ontologies is actually composed of concept and role hierarchies. See Table 3: 71%, 31%, 47%, 91% and 53 % of the rules in ontologies A, S, U, V and G, respectively, express atomic concept or role inclusions.

We can compile these sets of rules as preorders on predicates. The detailed presentation of how to compute and process these preorders is out of the scope of this paper. Briefly said, the preorders are integrated into the rewriting process, which allows to generate a smaller rewriting set, this set being unfolded at the end to produce the expected UCQ. Our purpose here is just to illustrate the fact that some improvements of the basic version can dramatically decrease the runtime, while still relying on the same fundamental mechanisms. Table 4 allows to compare these two versions: PURE[9] denotes the basic version of our tool and PURE$_H$ is the version with compiled hierarchical rules (note that compilation is performed offline, hence the algorithm takes as input the preorder and the non-hierarchical rules).

We also compared to two other query rewriting tools, Nyaya and Rapid. Nyaya is a tool dedicated to UCQ rewriting with linear and sticky existential rules, which implements the techniques presented in [GOP11], in particular an optimization for linear rules (which include DL-Lite ontologies). Table 4 shows that our tool is generally faster on the considered benchmark, even in its basic version, specially on Ontology A. This difference could be due to the fact that Nyaya does not directly process multiple-head rules, hence has to decompose them into atomic-head rules. For the large ontology G, Nyaya seemed to be still in a preprocessing step after several hours. Note that the very latest version of Nyaya includes parallel rewriting, which we did not consider here, since our tool does include this kind of optimization.

As far as we know, Nyaya is the only other tool able to process existential rules beyond lightweight DLs. We think that comparing to DL rewriting tools is not very relevant, since these systems make use of specific features, like predicate arity bounded by two, or the tree-model property. Tools tailored for DL-Lite exploit even further the very specific form of DL-Lite axioms. However, we compared to one of these tools, namely Rapid, to obtain an order of magnitude. Rapid is one of the fastest tools dedicated to DL-Lite ontologies [CTS11]. In Table 4, we can see that Rapid is indeed generally faster than our tool, the difference being less pronounced on the version with rule compilation.

Current work includes processing specific kinds of rules in a specific way, while

[9] Piece Unification based REwriting

| Rules | Query | PURE | PURE$_H$ | Nyaya | Rapid |
|---|---|---|---|---|---|
| A | Q1 | 170 | 120 | 1122 | 18 |
| | Q2 | 90 | 40 | 862 | 23 |
| | Q3 | 240 | 30 | 2363 | 34 |
| | Q4 | 440 | 200 | 5557 | 48 |
| | Q5 | 2100 | 440 | 33206 | 93 |
| S | Q1 | 0 | 0 | 4 | 7 |
| | Q2 | 10 | 10 | 4 | 9 |
| | Q3 | 40 | 40 | 46 | 13 |
| | Q4 | 90 | 20 | 7 | 12 |
| | Q5 | 110 | 80 | 8 | 15 |
| U | Q1 | 0 | 0 | 8 | 6 |
| | Q2 | 0 | 10 | 4 | 9 |
| | Q3 | 10 | 0 | 12 | 7 |
| | Q4 | 1370 | 120 | 6 | 13 |
| | Q5 | 20 | 10 | 10 | 15 |
| V | Q1 | 0 | 0 | 13 | 9 |
| | Q2 | 0 | 0 | 51 | 5 |
| | Q3 | 30 | 0 | 21 | 25 |
| | Q4 | 110 | 30 | 28 | 32 |
| | Q5 | 10 | 0 | 22 | 26 |
| G | Q1 | 10 | 0 | | 5 |
| | Q2 | 1090 | 620 | | 74 |
| | Q3 | 1050 | 290 | | 59 |
| | Q4 | 30 | 10 | | 10 |
| | Q5 | 1000 | 110 | | 40 |

Table 4: Runtime (ms) with several query rewriting tools

keeping a system able to process any set of existential rules. Other optimizations could be implemented, such as exploiting dependencies between rules to select the rules to be considered at each step. Moreover, the form of the considered output itself, i.e., a union of conjunctive queries, leads to combinatorial explosion. Considering semi-conjunctive queries instead of conjunctive queries as in [Tho13] can save much with respect to both the running time and the size of the output, without compromising the efficiency of query evaluation; in [Tho13] the piece-based rewriting operator is combined with query factorization techniques. We did not consider generating Datalog queries yet. Finally, further experiments should be performed on more complex ontologies. However, even if slightly more complex ontologies could be obtained by translation from description logics, real-world ontologies that would take advantage of the expressiveness of existential rules, as well as associated queries, are currently lacking.

**Acknowledgments.** We thank Giorgio Orsi for providing us with rule versions of ontologies A, S, U and V, as well as the version of Nyaya used for the experiments (October 2013 version). This work was partially funded by the ANR project PAGODA (ANR-12-JS02-007-01).

# References


[BLMS09] J.-F. Baget, M. Leclère, M.-L. Mugnier, and E. Salvat. Extending Decidable Cases for Rules with Existential Variables. In *IJCAI'09*, pages 677–682, 2009.

[BLMS11] J.-F. Baget, M. Leclère, M.-L. Mugnier, and E. Salvat. On Rules with Existential Variables: Walking the Decidability Line. *Artificial Intelligence*, 175(9-10):1620–1654, 2011.

[BV81] C. Beeri and M. Vardi. The implication problem for data dependencies. In *ICALP'81*, volume 115 of *LNCS*, pages 73–85, 1981.

[CGK08] A. Calì, G. Gottlob, and M. Kifer. Taming the Infinite Chase: Query Answering under Expressive Relational Constraints. In *KR'08*, pages 70–80, 2008.

[CGL+07] D. Calvanese, G. De Giacomo, D. Lembo, M. Lenzerini, and R. Rosati. Tractable reasoning and efficient query answering in description logics: The DL-Lite family. *J. Autom. Reasoning*, 39(3):385–429, 2007.

[CGL09] A. Calì, G. Gottlob, and T. Lukasiewicz. A general datalog-based framework for tractable query answering over ontologies. In *PODS'09*, pages 77–86, 2009.

[CGL12] A. Calì, G. Gottlob, and T. Lukasiewicz. A general Datalog-based framework for tractable query answering over ontologies. *J. Web Sem.*, 14, 2012.

[CGP10] A. Calì, G. Gottlob, and A. Pieris. Query Answering under Non-guarded Rules in Datalog+/-. In *RR'10*, pages 1–17, 2010.

[CLM81] A. K. Chandra, H. R. Lewis, and J. A. Makowsky. Embedded implicational dependencies and their inference problem. In *STOC'81*, pages 342–354. ACM, 1981.

[CLR03] A. Calì, D. Lembo, and R. Rosati. On the Decidability and Complexity of Query Answering over Inconsistent and Incomplete Databases. In *PODS'03*, pages 260–271, 2003.

[CM09] M. Chein and M.-L. Mugnier. *Graph-based Knowledge Representation and Reasoning—Computational Foundations of Conceptual Graphs*. Advanced Information and Knowledge Processing. Springer, 2009.

[CR12] C. Civili and R. Rosati. A Broad Class of First-Order Rewritable Tuple-Generating Dependencies. In *Datalog 2.0*, pages 68–80, 2012.

[CTS11] A. Chortaras, D. Trivela, and G. B. Stamou. Optimized Query Rewriting for OWL 2 QL. In *CADE'11*, pages 192–206, 2011.

[GHK+13] B. Cuenca Grau, I. Horrocks, M. Krötzsch, C. Kupke, D. Magka, B. Motik, and Z. Wang. Acyclicity notions for existential rules and their application to query answering in ontologies. *J. Artif. Intell. Res.*, 47:741–808, 2013.

[GOP11] G. Gottlob, G. Orsi, and A. Pieris. Ontological queries: Rewriting and optimization. In *ICDE'11*, pages 2–13, 2011.

[GS12] G. Gottlob and T. Schwentick. Rewriting Ontological Queries into Small Nonrecursive Datalog Programs. In *KR'12*, 2012.

[HN92] P. Hell and J. Nesetril. The core of a graph. *Discrete Mathematics*, 109(1-3):117–126, 1992.

[ISG12] M. Imprialou, G. Stoilos, and B. Cuenca Grau. Benchmarking Ontology-Based Query Rewriting Systems. In *AAAI'12*, 2012.

[KLMT12] M. König, M. Leclère, M.-L. Mugnier, and M. Thomazo. A Sound and Complete Backward Chaining Algorithm for Existential Rules. In *RR'12*, volume 7497, pages 122–138, 2012.

[KLMT13] M. König, M. Leclère, M.-L. Mugnier, and M. Thomazo. On the Exploration of the Query Rewriting Space with Existential Rules. In *RR'13*, pages 123–137, 2013.

[KLT+11] R. Kontchakov, C. Lutz, D. Toman, F. Wolter, and M. Zakharyaschev. The Combined Approach to Ontology-Based Data Access. In *IJCAI'11*, pages 2656–2661, 2011.

[KR11] M. Krötzsch and S. Rudolph. Extending Decidable Existential Rules by Joining Acyclicity and Guardedness. In *IJCAI'11*, pages 963–968, 2011.

[LMTV12] N. Leone, M. Manna, G. Terracina, and P. Veltri. Efficiently computable datalog programs. In *KR'12*, 2012.

[LTW09] C. Lutz, D. Toman, and F. Wolter. Conjunctive Query Answering in the Description Logic $\mathcal{EL}$ Using a Relational Database System. In *IJCAI'09*, pages 2070–2075, 2009.

[Mug11] M.-L. Mugnier. Ontological Query Answering with Existential Rules. In *RR'11*, pages 2–23, 2011.

[OP11] G. Orsi and A. Pieris. Optimizing query answering under ontological constraints. *PVLDB*, 4(11):1004–1015, 2011.

[OWL09] W3C OWL Working Group. *OWL 2 Web Ontology Language: Document Overview*. W3C Recommendation, 2009.

[PUHM09] H. Pérez-Urbina, I. Horrocks, and B. Motik. Efficient Query Answering for OWL 2. In *ISWC'09*, pages 489–504, 2009.

[RA10] R. Rosati and A. Almatelli. Improving query answering over DL-Lite ontologies. In *KR'10*, 2010.

[RMC12] M. Rodriguez-Muro and D. Calvanese. High Performance Query Answering over DL-Lite Ontologies. In *KR'12*, 2012.

[SM96] E. Salvat and M.-L. Mugnier. Sound and Complete Forward and Backward Chainings of Graph Rules. In *ICCS'96*, pages 248–262, 1996.

[TBMR12] M. Thomazo, J.-F. Baget, M.-L. Mugnier, and S. Rudolph. A generic querying algorithm for greedy sets of existential rules. In *KR'12*, 2012.

[Tho13] M. Thomazo. Compact Rewriting for Existential Rules. In *IJCAI'13*, 2013.

[VSS14] T. Venetis, G. Stoilos, and G. B. Stamou. Query Extensions and Incremental Query Rewriting for OWL 2 QL Ontologies. *J. Data Semantics*, 3(1):1–23, 2014.

# Appendix: Proof of Theorem 17

To prove the completeness of the single-piece based operator, we first prove the following property:

**Property 24** *For any piece-unifier $\mu$ of $Q$ with $R$, there is a sequence of rewritings of $Q$ with $R$ using exclusively most general single-piece unifiers and leading to a BCQ $Q^s$ such that $Q^s \geq \beta(Q, R, \mu)$.*

*Proof:* We first introduce some notations. Given a partition $P$ and $x$ a term occurring in $P$, $P(x)$ is the class of $P$ that contains $x$. Let $P$ and $P'$ be two partitions such that the terms of $P'$ are included in the terms of $P$ and any class of $P'$ is included in a class of $P$: then we say that $P'$ is a subpart of $P$ (note that if $P'$ and $P$ are defined on the same set, it means that $P'$ is finer than $P$)

Let $Pc_1, \ldots, Pc_n$ be the pieces of $Q'$ according to $\mu = (Q', H', P_u)$ and let $u$ be a substitution associated to $P_u$. Let $Q_0 = Q, Q_1, \ldots Q_n = Q^s$ be a sequence of rewritings of $Q$ built as follows: for $1 \leq i \leq n$, $Q_i = \beta(Q_{i-1}, R_i, \mu_i)$ where $\mu_i = (Q'_i, H'_i, P^i_u)$ and $u_i$ is a substitution associated with $P^i_u$ with:

- $R_i$ is a safely renamed copy of $R$ by a variable renaming $h_i$.
- $H'_i$ is the image by $h_i$ of the subset of $H'$ unified by $u$ with $Pc_i$
- $P^i_u$ is obtained from partition $h_i(P_u)$ (built from $P_u$ by applying $h_i$) by (1) restricting it to the terms of $Q'_i$ and $H'_i$ (2) refining it as much as possible while keeping the property that it is associated with a unifier of $H'_i$ and $Q'_i$. Note that $P^i_u$ is a subpart of $h_i(P_u)$.
- Let $u^\circ_i = u_i \circ u_{i-1} \circ \ldots \circ u_1$. Let $P^{i\circ}_u$ be the partition assigned to $u^\circ_i$. We know that $P^{i\circ}_u$ is the join of $P^1_u, \ldots P^i_u$, thus $P^{i\circ}_u$ is a subpart of $P^h_u$, the join of the $h_i(P_u)$ for $1 \leq i \leq n$. Indeed, for each $i$, $P^i_u$ is a subpart of $h_i(P_u)$ and the following property is easily checked: let $s_1$ and $s_2$ be substitutions with *disjoint* domains, and $P^1_s$, $P^2_s$ be their associated partitions; then, the partition assigned to $s_1 \circ s_2$ (and to $s_2 \circ s_1$) is exactly the join of $P^1_s$ and $P^2_s$.
- $Q'_1 = Pc_1$ and for $i > 1$, $Q'_i = u^\circ_{i-1}(Pc_i)$. We ensure the property than $\forall i$, $u^\circ_{i-1}(Pc_i) \cap u^\circ_{i-1}(Q \setminus Q') = \emptyset$. If $u^\circ_{i-1}(Pc_i) \cap u^\circ_{i-1}(Q \setminus Q') \neq \emptyset$, we remove $\mu_i$ from the sequence because it is useless since $u^\circ_{i-1}(Pc_i) \subseteq u^\circ_{i-1}(Q \setminus Q')$. Indeed, let $a \in u^\circ_{i-1}(Pc_i) \cap u^\circ_{i-1}(Q \setminus Q')$, there are $b \in Pc_i$ and $b' \in Q \setminus Q'$, $b \neq b'$ such that $u^\circ_{i-1}(b) = u^\circ_{i-1}(b') = a$, so $terms(b) \subseteq sep(Pc_i)$, so $\{b\}$ is a piece, so $Pc_i = \{b\}$ and then $u^\circ_{i-1}(Pc_i) = \{a\} \subseteq u^\circ_{i-1}(Q \setminus Q')$. For similar reasons, we ensure the property that $\forall i, \forall j > i$, $u^\circ_{i-1}(Pc_i) \cap u^\circ_{i-1}(Pc_j) = \emptyset$.

We now show that:

1. $\mu_i$ is a piece-unifier
2. $\mu_i$ is a most general piece-unifier
3. $\mu_i$ is a single-piece unifier

For the first point:

- $Q'_i \subseteq Q_{i-1}$ since $\forall i$, $u^\circ_{i-1}(Pc_i) \cap u^\circ_{i-1}(Q \setminus Q') = \emptyset$ and $\forall i, \forall j > i$ $u^\circ_{i-1}(Pc_i) \cap u^\circ_{i-1}(Pc_j) = \emptyset$

- $H'_i \subseteq \mathit{head}(R_i)$ by construction.
- $P^i_u$ satisfies the conditions of a piece-unifier because $P_u$ satisfies them and $P^i_u$ is a subpart of $h_i(P_u)$.

For the second point, since $P^i_u$ is the finest partition associated with a piece-unifier of $H'_i$ and $Q'_i$, we are sure that $\mu_i$ is a most general piece-unifier.

For the third point, note that each atom of $Q'_i$ corresponds to at least one atom of $Pc_i$. Thus if $Pc_i$ is composed of a unique atom, so is $H'_i$ which thus forms a single-piece. Otherwise, $Pc_i$ is a single-piece from more than one atom; each atom $a$ of $Pc_i$ contains a variable $x$ such that $P_u(x)$ contains an existential variable $y$ which comes from the subset of $H'$ unified by $u$ with $Pc_i$. Thus the corresponding atom $u^\circ_{i-1}(a)$ in $Q'_i$ is such that $P^i_u(u^\circ_{i-1}(x))$ contains the existential variable $h_i(y)$. So $Q'_i$ forms a single piece.

At the end of the sequence, $Q_n \subseteq u^\circ_n(Q \setminus Q') \cup \bigcup_{j\in 1..n}(u_n(\dots u_j(\mathit{body}(R_j))))$ and the terms of $P^{n\circ}_u$ are the same as the terms of $P^h_u$. Since $P^{n\circ}_u$ is a subpart of $P^h_u$, we can say that $P^{n\circ}_u$ is finer than $P^h_u$ so, there is a substitution $s$ such that $u^h = s \circ u^\circ_n$ and $s(u^\circ_n(Q \setminus Q')) = u^h(Q \setminus Q')$. Let $h$ be the substitution obtained by making the union of the inverses of the $h_i$, then $h(u^h(Q \setminus Q') = u(Q \setminus Q')$, so $h \circ s$ is a homomorphism from $u^\circ_n(Q \setminus Q')$ to $u(Q \setminus Q')$. Then we can prove that for all $j$, $1 \leq j \leq n$, $h(s(u_n(\dots u_j(\mathit{body}(R_j))))) = u(\mathit{body}(R))$. Indeed, $u_n(\dots u_j(\mathit{body}(R_j))) = u_n(\dots u_1(\mathit{body}(R_j)))$ since the terms of $\mathit{body}(R_j)$ do not appear in $u_i$ ($i < j$).

To conclude the proof, we have $h(s(Q_n)) \subseteq u(\mathit{body}(R)) \cup u(Q \setminus Q') = \beta(Q, \mu, R)$, hence $h \circ s$ is a homomorphism from $Q_n$ to $\beta(Q, \mu, R)$, thus $Q_n \geq \beta(Q, \mu, R)$. □

**Theorem 17** *Given a BCQ $Q$ and a set of rules $\mathcal{R}$, the set of rewritings of $Q$ obtained by considering exclusively most general single-piece unifiers is sound and complete.*

*Proof:* Soundness holds trivially since a single-piece unifier is a piece-unifier.
For completeness, thanks to Theorem 12, we just have to show by induction on $k$, the length of the rewriting sequence leading from $Q$ to a $k$-piece-rewriting of $Q$, that: for any $k$-piece-rewriting $Q^r$ of $Q$, there exists $Q^s$ a piece-rewriting of $Q$ obtained by using exclusively most general single-piece unifiers such that $Q^s \geq Q^r$.
For $k = 0$ the property is trivially satisfied.
For $k \geq 1$, one has $Q^r = \beta(Q^{r'}, R, \mu)$, with $Q^{r'}$ being a piece-rewriting of $Q$ obtained by a piece-rewriting sequence of length $k - 1$. By induction hypothesis, there exists $Q^{s'}$ a piece-rewriting of $Q$ obtained by using exclusively single-piece unifiers such that $Q^{s'} \geq Q^{r'}$. By Lemma 15, either $Q^{s'} \geq Q^r$, or there is a piece-unifier $\mu'$ of $Q^{s'}$ with $R$ such that $\beta(Q^{s'}, R, \mu') \geq Q^r$. In this latter case, thanks to Property 24, there is a sequence of rewritings of $Q^{s'}$ with $R$ using only single-piece unifiers and leading to a CQ $Q^s$ such that $Q^s \geq \beta(Q^{s'}, R, \mu')$. □