On Rules with Existential Variables: Walking the Decidability Line[☆]

Jean-François Baget[a], Michel Leclère[b], Marie-Laure Mugnier[*,b], Eric Salvat[c]

[a] *INRIA Sophia Antipolis, France*
[b] *University of Montpellier 2, France*
[c] *IMERIR, France*

Abstract

We consider positive rules in which the conclusion may contain existentially quantified variables, which makes reasoning tasks (such as conjunctive query answering or entailment) undecidable. These rules, called $\forall\exists$-rules, have the same logical form as tuple-generating dependencies in databases and as conceptual graph rules. The aim of this paper is to provide a clearer picture of the frontier between decidability and non-decidability of reasoning with these rules. Previous known decidable classes were based on forward chaining. On the one hand we extend these classes, on the other hand we introduce decidable classes based on backward chaining. A side result is the definition of a backward mechanism that takes the complex structure of $\forall\exists$-rule conclusions into account. We classify all known decidable classes by inclusion. Then, we study the question of whether the union of two decidable classes remains decidable and show that the answer is negative, except for one class and a still open case. This highlights the interest of studying interactions between rules. We give a constructive definition of dependencies between rules and widen the landscape of decidable classes with conditions on rule dependencies and a mixed forward/backward chaining mechanism. Finally, we integrate rules with equality and negative constraints to our framework.

Key words: Rules, TGD, Decidability, Forward Chaining, Chase, Backward Chaining, Rule Dependency

To appear in Artificial Intelligence

1. Introduction

Rules are fundamental constructs in knowledge-based systems and databases. Here we consider positive rules in first-order logic (FOL) without functions, of the form $H \rightarrow C$, where H and C are conjunctions of atoms, respectively called the hypothesis

[☆] This is an extended version of two papers published at IJCAI 2009 [7] and KR 2010 [5], respectively.

[*] Corresponding author.

Email addresses: baget@lirmm.fr (Jean-François Baget), leclere@lirmm.fr (Michel Leclère), mugnier@lirmm.fr (Marie-Laure Mugnier), salvat@imerir.com (Eric Salvat)

and conclusion of the rule, and there might be *existentially* quantified variables in the conclusion. E.g., the rule $R = \mathit{Human}(x) \rightarrow \mathit{Parent}(y, x) \wedge \mathit{Human}(y)$ stands for the formula $\forall x(\mathit{Human}(x) \rightarrow \exists y(\mathit{Parent}(y, x) \wedge \mathit{Human}(y)))$. We call this kind of rules $\forall\exists$-rule (pronounced "forall-exist-rule"). Existentially quantified variables in the conclusion, associated with arbitrary complex conjunctions of atoms, make $\forall\exists$-rules very expressive but also lead to undecidability of reasoning. Several decidable classes have been exhibited, in both artificial intelligence and database domains. The general aim of this paper is to bring out a clearer picture of the frontier between decidability and undecidability of reasoning.

$\forall\exists$-rules have the same logical form as very general database dependencies called tuple-generating dependencies (TGD) [1] and as conceptual graph (CG) rules [36, 8]. TGDs have been extensively used in databases as high-level generalizations of different kinds of constraints. An example of a prominent application is data exchange, where a building block is the notion of schema mapping, which specifies the relationships between heterogeneous database schemas [30]. Schema mappings rely on TGDs to generate unknown values, i.e., existentially quantified variables. Among new applications, let us mention web data extraction, which consists of automatically identifying and extracting objects from web pages using domain-specific knowledge, and generating from them higher level objects, which involves dynamically creating new object identifiers [15]. More generally, $\forall\exists$-rules are well-suited to applications where new entities need to be automatically generated.

Querying knowledge bases is a central problem in knowledge representation and in database theory. A knowledge base (KB) is classically composed of a terminological part (called here the ontology) and an assertional part (called here the facts). KB queries are supposed to be at least as expressive as the basic queries in databases, i.e., conjunctive queries, which can be seen as existentially closed conjunctions of atoms. A fundamental decision problem is thus (Boolean) conjunctive query answering, which can be expressed as an entailment problem: is a (Boolean) conjunctive query entailed by a KB? $\forall\exists$-rules are an abstraction particularly well-suited to the representation of ontological knowledge in this context. They generalize several specific knowledge representation languages adapted to query answering, e.g., RDFS [31] (the basic semantic web language), constraints in F-logic-Lite [16, 12] (a powerful subset of F-logic, a formalism for object-oriented deductive databases), as well as the core of new families of description logics (DL) tailored for conjunctive query answering [18, 33, 6, 13].

In this paper, instead of focusing on a particular formalism, we consider an abstract framework expressed in first-order logic and based on $\forall\exists$-rules. Our fundamental entailment problem (noted ENTAILMENT) can be expressed as the above mentioned Boolean conjunctive query answering problem, or equivalently as a fact entailment problem (is a fact entailed by a KB?) or as a rule entailment problem (is a rule entailed by a KB?). These three problems can also be recast as fundamental problems in databases, respectively known as (Boolean) conjunctive query answering under constraints expressed by TGDs, conjunctive query containment w.r.t. a set of TGDs and TGD implication. Our results on ENTAILMENT are thus directly applicable to any of these problems.

As ENTAILMENT is undecidable ([10][19] for TGDs, [3][8] for CG rules), our aim is to define large decidable classes of $\forall\exists$-rules, expressive enough to generalize various

specific formalisms or languages for describing ontologies. Until now, there were only decidable cases based on the *forward chaining* scheme (called the *chase* in databases [32]). Forward chaining may not halt, as can be seen for instance with the rule $R = \mathit{Human}(x) \rightarrow \mathit{Parent}(y, x) \wedge \mathit{Human}(y)$: once R has been applied, it can be applied again infinitely and each application produces a fact that is not equivalent to any of the previous ones. Exhibited decidable classes of rules are based on cases where the forward chaining halts (e.g., positive Datalog rules, in which the rule conclusions do not add new variables), or can be stopped after a number of steps depending on the KB and the query [32, 12]. We enrich the landscape of decidable cases by extending known decidable classes based on forward chaining but also by introducing new decidable classes based on backward chaining.

The next question is whether known decidable cases can be combined while keeping decidability. We show that the answer is generally "no" if by "combining" we mean making the rough union of decidable sets of rules. We thus refine this notion by considering the notion of dependency between rules introduced in [4]. We use the structure of the graph that encodes rule dependencies to define conditions under which decidable classes of rules can be safely combined.

Outline of our contributions. We now present our main results in further detail. Decidable classes of $\forall\exists$-rules have long been defined and used. To classify them, we distinguish between *abstract* and *concrete* decidable classes. Abstract classes are based on the behavior of reasoning mechanisms, i.e., forward and backward chaining mechanisms. This behavior is generally not provided with a finite procedure allowing to determine whether a given set of rules has the property or not. Concrete classes are defined by computable syntactic properties.

In this paper, we identify three *abstract classes*. Two of them are based on a forward chaining scheme: *finite expansion sets* (*fes*) [8], ensuring that a finite number of rule applications is sufficient to answer any query, and the more general *bounded treewidth sets* (*bts*), inspired by the work of [12], that relies on the finite treewidth model property of [25]. The third class is based on a backward chaining scheme: a *finite unification set* (*fus*) [7] ensures that any query can be finitely rewritten. Unsurprisingly, these abstract classes are not recognizable, i.e., checking whether a given set of rules belongs to one of these classes is undecidable (Theorem 5).

Since abstract classes are not recognizable, we turn our attention to *concrete classes* implementing their abstract behavior. These classes are less expressive but recognizable.

In relationship with forward chaining, we introduce two new decidable classes implementing *bts* behavior: *frontier-guarded* rules and their extension to *weakly frontier-guarded* sets of rules, which are generalizations of the classes defined in [12]. These classes have the advantage of unifying some other known classes [7]. We point out that their expressive power allows us to represent a set of description logic statements that are particularly interesting in the context of new DLs designed for query answering. To show that (weakly) frontier-guarded rules have the *bts* property, we introduce a simple tool, called the *Derivation Graph*, as well as reduction operations on this graph. The fundamental property of this tool is as follows: if every derivation graph produced by a set of rules can be reduced to a tree (or a forest), then this set of rules has the *bts*

property, which is especially the case for (weakly) frontier-guarded rules (Theorem 7).

To study the backward chaining behavior, we define a backward chaining mechanism tailored for $\forall\exists$-rules. The backward chaining mechanisms classically used in logic programming process rules and goals (i.e., queries) atom by atom. Our backward chaining mechanism keeps accounting for the complex structure of a rule conclusion induced by existential variables, by characterizing sets of atoms which should not be processed separately. More precisely, rule conclusions and goals are decomposed into subsets of atoms, called *pieces*, which can be seen as "units of knowledge" and are processed as a whole. E.g., in the above rule R, $\{\mathit{Parent}(y, x), \mathit{Human}(y)\}$ is a piece. We thus define unification based on pieces, called *piece-unification*. We exhibit two concrete classes implementing *fus* behavior: *atomic-hypothesis* rules and *domain-restricted* rules.

Having a range of decidable classes at our disposal, an interesting question is whether the union of two decidable classes remains decidable. This question is of utmost importance if we want to merge two ontologies for which decidability of reasoning is ensured by different syntactic properties, or if, having implemented the semantics of two knowledge representation languages with sets of rules belonging to decidable classes, we want to consider the language built from the union of both languages. We present a systematic study of this question for all decidable classes we are aware of. With the exception of *disconnected rules*, which are universally compatible, and of a still open case, we show that the union of two incomparable decidable classes is never decidable (Theorem 13). These rather negative results on the rough union of decidable cases highlight the interest of precisely studying interactions between rules.

A complementary result is that ENTAILMENT remains undecidable even with a *single rule* (Theorem 8). This result has an important immediate consequence: adding a single rule to *any* set belonging to a decidable class of rules can make the problem undecidable.

We then turn our attention to dependencies between rules [4]: a rule R' is said to *depend* on a rule R if the application of R on a fact may trigger a *new* application of R'. We show that this abstract definition can be effectively implemented by the piece-unification of our backward chaining mechanism: R' depends on R if and only if there is a piece-unifier between the hypothesis of R' and the conclusion of R (Theorem 15). We are thus able to effectively build a graph encoding dependencies between rules.

ENTAILMENT is decidable when this graph has no circuit [4]. Furthermore, when all strongly connected components of this graph are *fes* (resp. *fus*), then the set of rules is a *fes* (resp. *fus*), Theorem 17. Even more interesting is the fact that, with additional conditions on this graph, it is possible to combine a *bts* and a *fus* into a new decidable class, which strictly contains both *bts* and *fus* (Theorem 19). In the case where this *bts* is a *fes*, or more generally is provided with an effective procedure based on forward chaining, one can use a mixed forward/backward chaining algorithm. Note that this algorithm does not merge forward chaining and backward chaining into a single mechanism (like some Datalog evaluation techniques [1]), but rather partitions the set of rules into a *bts* part and a *fus* part, and processes each part separately: the *bts* in forward chaining and the *fus* in backward chaining.

This combination of abstract classes effectively combines any concrete classes implementing their behavior, including those we are not yet aware of. This shows that

using abstract classes is a powerful method for building generic decidability results.

Finally, we consider the extension of the framework with rules with equality and negative constraints. Whereas negative constraints come for free, equality is a well-known source of undecidability. In particular, we show that the addition of one equality rule to a *fes* (*a fortiori* a *bts*) or a *fus* does not preserve decidability (Theorem 20).

Paper Organization. Section 2 defines the basic framework. Section 3 introduces backward chaining based on pieces. Sections 4 and 5 respectively deal with abstract and concrete decidable cases. Section 6 studies the union of decidable cases. Section 7 introduces the graph of rules dependencies and uses it to safely combine decidable cases. Rules with equality and negative constraints are considered in Section 8. Section 9 is devoted to related work in databases and conceptual graphs.

2. Preliminaries

In this section,we provide fundamental definitions and properties on facts, $\forall\exists$-rules and the associated entailment problems. In the fragment of facts, entailment can be checked by a homomorphism test. In the fragment of facts and rules, it can be computed by a forward chaining mechanism based on homomorphism. The associated soundness and completeness results (Theorems 1 and 2) are not new: they follow from early results in databases (resp. [20] and [34][2]), as well as from more recent results in a graph-based framework (resp. [22] and [35]). However, we provide novel and simple proofs, whose intermediate results will be used as building blocks for the remainder of the paper. These proofs are detailed in Appendix A.

2.1. Vocabulary

We consider first-order logical languages with constants but no other function symbols. A *term* is thus a variable or a constant. A *vocabulary* $\mathcal{V} = (\mathcal{P}, \mathcal{C})$ is composed of two disjoint sets: a set $\mathcal{P}$ of *predicates* and a set $\mathcal{C}$ of *constants*. Hence, an atom on $\mathcal{V}$ is of form $p(t_1, \ldots, t_k)$, where p is a predicate in $\mathcal{P}$ with arity k and the t_i are variables or constants in $\mathcal{C}$. A *ground* atom contains only constants. In examples, we use uppercase letters for constants and lowercase letters for variables. Given a formula ϕ, we note $\mathcal{V}(\phi)$ the restriction of $\mathcal{V}$ to symbols occurring in ϕ and $pred(\phi)$, $const(\phi)$, $var(\phi)$, $term(\phi)$ resp. the set of predicates, constants, variables and terms occurring in ϕ.

Definition 1 (Interpretation of a vocabulary). *An* interpretation *of a vocabulary* $\mathcal{V} = (\mathcal{P}, \mathcal{C})$ *is a pair* $I = (\Delta, .^I)$ *where* Δ *is a (possibly infinite) set called the* interpretation domain *and* $.^I$ *is an* interpretation function *such that:*

1. *for each constant* $c \in \mathcal{C}$, $c^I \in \Delta$;
2. *for each predicate of arity* k $p \in \mathcal{P}$, $p^I \subseteq \Delta^k$;
3. *for each pair* (c, c') *of distinct constants in* $\mathcal{C}$, $c^I \neq c'^I$.

The third condition in the above definition corresponds to the *unique name assumption*, which is often made in knowledge representation. However, note that as long as equality is not considered (see Section 8), adopting the unique name assumption or not does not make any difference in the considered reasoning tasks.

We rely on the classical definitions to assert that a formula ϕ is true in an interpretation I of $\mathcal{V}(\phi)$, i.e., I is a *model* of ϕ. $\models$ denotes the classical logical consequence (or entailment) and $\equiv$ the associated equivalence.

2.2. Facts

We now introduce the notions of fact and homomorphism. In the literature, a fact is classically defined as a ground atom; since $\forall\exists$-rules produce atoms with variables that are globally existentially quantified, we extend the notion of fact to an existentially closed conjunction of atoms.

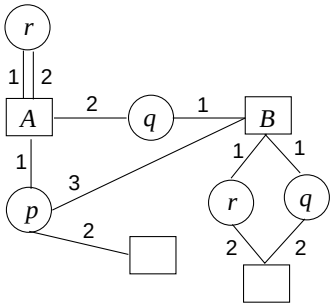
Definition 2 (Conjuncts, Facts). *Given a vocabulary $\mathcal{V} = (\mathcal{P}, \mathcal{C})$, a* conjunct *on $\mathcal{V}$ is a conjunction of atoms on $\mathcal{V}$. A* fact *on $\mathcal{V}$ is the existential closure of a conjunct on $\mathcal{V}$.*

By default, we assume that conjuncts and facts are *finite*. For some results, we will consider possibly infinite conjuncts or facts (more precisely, we will have to consider homomorphisms from finite facts to possibly infinite facts). W.l.o.g. we exclude duplicate atoms in facts, which allows to see a fact as a *set of atoms*. For instance the fact $F = \exists x \exists y (p(A, x, B) \wedge q(B, y) \wedge r(B, y) \wedge q(B, A) \wedge r(A, A) \wedge r(A, A))$ can be seen as the set $\{p(A, x, B), q(B, y), r(B, y), q(B, A), r(A, A)\}$. In proofs, we will use a specific fact of which every fact is a consequence:

Definition 3 (*All-true* fact). *The* all-true *fact on a finite vocabulary $\mathcal{V} = (\mathcal{P}, \mathcal{C})$ is the fact containing all atoms that can be built with the set of predicates $\mathcal{P}$ and the set of terms $\mathcal{C}$ if $\mathcal{C} \neq \emptyset$, otherwise $\{x\}$ where x is a variable.*

A useful encoding of a fact F is its encoding as a *directed labeled multiple hypergraph*, say $\mathcal{F}$: the sets of nodes and hyperarcs in $\mathcal{F}$ are respectively in bijection with $term(F)$ and with the set of atoms F; this hypergraph is said to be multiple because there may be several hyperarcs with the same argument list (but with different predicates); to fully encode a fact, nodes and hyperarcs are labeled: a node in $\mathcal{F}$ assigned to a constant is labeled by this constant, otherwise it is not labeled (indeed, variable names are not needed to encode the fact) and a hyperarc in $\mathcal{F}$ is labeled by the corresponding predicate.

For drawing purposes, it is convenient to consider the *incidence graph* of $\mathcal{F}$: it is a bipartite undirected multigraph (i.e., multiple graph), with one set of nodes representing the terms (i.e., the nodes in $\mathcal{F}$), and the other set of nodes representing the atoms (i.e., the hyperarcs in $\mathcal{F}$). More precisely, for each atom $p(t_1, \ldots, t_k)$ in F, there is a node labeled by p and this node is incident to k edges linking it to the nodes assigned to $t_1, \ldots, t_k$. Each edge is labeled by the position of the corresponding term in the atom. See Figure 1. Note that this graph can be seen as the basic conceptual graph assigned to the formula [23]. The hypergraph/graph view of facts enables one to focus on their structure. For instance in this paper we rely on it to define the decidable class of "bounded treewidth sets" (Section 4) and it is at the origin of the piece notion (Section 3.1).



$\{p(A, x, B), q(B, y), r(B, y), q(B, A), r(A, A)\}$

Figure 1: Graph view of a fact

Definition 4 (Substitution, Homomorphism, Isomorphism). *Given a set of variables X and a set of terms T, a* substitution *σ of X by T (notation $\sigma : X \to T$) is a mapping from X to T. Given a conjunct C, $\sigma(C)$ denotes the conjunct obtained from C by replacing each occurrence of $x \in X \cap var(C)$ by $\sigma(x)$. If a fact F is the existential closure of a conjunct C, then $\sigma(F)$ is the existential closure of $\sigma(C)$. A* renaming substitution *is an injective substitution that maps variables to "fresh" variables*[1]*. A* homomorphism *from a fact F to a (possibly infinite) fact F' is a substitution σ of $var(F)$ by (a subset of) term(F') such that $\sigma(F) \subseteq F'$. If there is a homomorphism from F to F', we say that F* maps to *F'. An* isomorphism *from a fact F to a fact F' is a bijective substitution σ from var(F) to var(F') such that $\sigma(F) = F'$.*

For convenience, we will sometimes extend the domain of a substitution to a set of terms, with a constant being necessarily mapped to the same constant. The conjunction of two facts F_1 and F_2 is equivalent to a fact, say F; in the set-representation of facts, F is obtained by making the union of F_1 and $\sigma(F_2)$, where σ is a renaming substitution of variables common F_1 and F_2. In the following, we identify a set of facts with a single fact.

Given a fact F and a substitution σ, we will often consider a *safe substitution in F according to σ*, denoted $\sigma^{\text{safe}}(F)$. This substitution replaces all variables x of F that are in the domain of σ by $\sigma(x)$ and renames all other variables, i.e., $\sigma^{\text{safe}}(F) = \sigma(\sigma'(F))$, where σ' is a renaming substitution of *term*$(F) \setminus$ *domain*(σ).

Homomorphism and isomorphism can also be defined on the hypergraphs or multigraphs corresponding to facts (f.i., [23]). The next theorem expresses that homomorphism checking is sound and complete in the logical fragment of facts, which has been proven in several contexts:

Theorem 1 (Homomorphism). *Let F and F' be two facts, with F' being possibly infinite. $F' \models F$ if and only if there is a homomorphism from F to F'.*

[1] A fresh variable x is an element of a totally ordered infinite set of variables $\mathcal{V}_f$, that is disjoint from the set of variables used in the input knowledge base, and such that x is greater than all elements of $\mathcal{V}_f$ already introduced.

Proof: See Appendix A.

□

Let us say that two facts F and F' are *hom-equivalent* if they map to each other by homomorphism. From the previous theorem, it holds that F and F' are *hom-equivalent* if and only if $F \equiv F'$. Thus, from now on we identify both notions. The following notion of a core comes from graph theory:

Definition 5 (Core). *The core of a fact F, denoted core(F), is a minimal subset of F equivalent to F.*

The following properties are folklore: the core of a fact is unique up to isomorphism; given two facts F and F', if $F \equiv F'$ then *core*(F) and *core*(F') are isomorphic.

2.3. Rules

We now introduce $\forall\exists$-rules and define the saturation mechanism, which is at the core of a breadth-first forward chaining mechanism, known as the *chase* in databases [32].

Definition 6 ($\forall\exists$-rule, Frontier). *A* $\forall\exists$-rule $R = (H, C)$ *on a vocabulary $\mathcal{V}$ is a closed formula of form $\forall x_1 \dots \forall x_p(H \rightarrow (\exists z_1 \dots \exists z_q C))$ where H and C are two (finite) conjuncts on $\mathcal{V}$; $\{x_1, \dots, x_p\} = var(H)$; and $\{z_1 \dots z_q\} = var(C) \setminus var(H)$. H and C are respectively called the* hypothesis *and the* conclusion *of R, also noted hyp(R) and conc(R). The* frontier *of R (notation fr(R)) is the set of variables occurring in both H and C: fr$(R) = var(H) \cap var(C)$.*

In examples, we use the form $R = H \rightarrow C$ with implicit quantifiers. In the following, we will often consider H and C as facts by considering their existential closure.

Note that a $\forall\exists$-rule is not a Horn clause because of existential variables in its conclusion. However, both are closely related, since by skolemisation (i.e., replacing each existential variable by a Skolem function) a $\forall\exists$-rule can be transformed into a set of Horn clauses with functions (e.g., see Example 4 in Section 3.2). This transformation yields a reduction from our entailment problem (see Section 2.4) to the entailment problem on a set of Horn clauses.

Definition 7 (Application of a $\forall\exists$-rule). *Let F be a fact and $R = (H, C)$ be a $\forall\exists$-rule. R is said* applicable *to F if there is a homomorphism, say π, from H to F. In that case, the* application of R to F according to π *produces a fact $\alpha(F, R, \pi) = F \cup \pi^{\text{safe}}(C)$. $\alpha(F, R, \pi)$ is said to be an* immediate derivation *from F. This rule application is said to be* redundant *if $\alpha(F, R, \pi) \equiv F$.*

Note that $\alpha(F, R, \pi)$ is unique up to isomorphism (encoded in the safe substitution according to π). To check whether $\alpha(F, R, \sigma) \equiv F$, it suffices to check that $\alpha(F, R, \sigma)$ maps to F.

Definition 8 (Derivation). *Let F be a fact and $\mathcal{R}$ be a set of $\forall\exists$-rules. A fact F' is called an* $\mathcal{R}$-derivation *of F if there is a finite sequence (called the* derivation sequence*) $F = F_0, F_1, \ldots, F_k = F'$ such that for all $1 \leq i \leq k$, there is a rule $R = (H, C) \in \mathcal{R}$ and a homomorphism π from H to F_{i-1} with $F_i = \alpha(F_{i-1}, R, \pi)$, i.e., F_i is an immediate derivation from F_{i-1}.*

Intuitively, the saturation mechanism can be seen as a breadth-first forward chaining scheme. Let F_0 be the initial fact F. Each step consists of producing a fact, say F_i at step i, from the current fact F_{i-1}, by computing all homomorphisms from each rule hypothesis to F_{i-1}, then performing all corresponding rule applications. The fact F_k obtained after the step k is called the k-saturation of F.

Definition 9 (k-saturation). *Let F be a fact and $\mathcal{R}$ be a set of $\forall\exists$-rules. $\Pi(\mathcal{R}, F)$ denotes the set of homomorphisms from a rule hypothesis in $\mathcal{R}$ to F:*
$\Pi(\mathcal{R}, F) = \{(R, \pi) | R = (H, C) \in \mathcal{R}$ *and* π *is a homomorphism from* H *to* $F\}$.
The direct saturation of F with $\mathcal{R}$ is defined as:
$\alpha(F, \mathcal{R}) = F \cup_{(R=(H,C),\pi)\in\Pi(\mathcal{R},F)} \pi^{\text{safe}}(C)$.
The k-saturation of F with $\mathcal{R}$ is denoted by $\alpha_k(F, \mathcal{R})$ and is inductively defined as follows: $\alpha_0(F, \mathcal{R}) = F$ and, for $i > 0$, $\alpha_i(F, \mathcal{R}) = \alpha(\alpha_{i-1}(F, \mathcal{R}), \mathcal{R})$.

We note $\alpha_\infty(F, \mathcal{R}) = \cup_{k\in\mathbb{N}}\alpha_k(F, \mathcal{R})$. $\alpha_\infty(F, \mathcal{R})$ is possibly infinite. A variant of k-saturation (let us note it α^c_k) is obtained by computing the core of the obtained fact at each step: $\alpha^c_0(F, \mathcal{R}) = \mathit{core}(F)$ and, for $i > 0$, $\alpha^c_i(F, \mathcal{R}) = \mathit{core}(\alpha(\alpha^c_{i-1}(F, \mathcal{R}), \mathcal{R}))$. We note $\alpha^c_\infty(F, \mathcal{R}) = \cup_{k\in\mathbb{N}}\alpha^c_k(F, \mathcal{R})$. A straightforward induction allows to check that, for any $i > 0$, $\mathit{core}(\alpha_i(F, \mathcal{R}))$ is isomorphic to $\alpha^c_i(F, \mathcal{R})$, i.e., $\alpha_i(F, \mathcal{R}) \equiv \alpha^c_i(F, \mathcal{R})$.

Property 1. *Let F and F' be two facts and $\mathcal{R}$ be a set of $\forall\exists$-rules. There is a homomorphism from F' to $\alpha_\infty(F, \mathcal{R})$ if and only if there is an integer k and a homomorphism from F' to $\alpha_k(F, \mathcal{R})$.*

Proof: ($\Leftarrow$) Trivial, since $\alpha_k(F, \mathcal{R}) \subseteq \alpha_\infty(F, \mathcal{R})$. ($\Rightarrow$) We number each atom a in $F^* = \alpha_\infty(F, \mathcal{R})$ by the rank at which it has been produced, i.e., by the smallest integer i such that $a \in \alpha_i(F, \mathcal{R})$. Suppose that there is a homomorphism π from F' to F^*. Since $\pi(F')$ is a finite subset of atoms of F^*, these atoms admit a maximum rank, say k. Then there is a homomorphism from F' to $\alpha_k(F, \mathcal{R})$. □

Note that the above property also holds for the variant of k-saturation based on the core computation: there is a homomorphism from F' to $\alpha^c_\infty(F, \mathcal{R})$ if and only if there is an integer k and a homomorphism from F' to $\alpha^c_k(F, \mathcal{R})$.

Property 2. *Let F and F' be two facts and $\mathcal{R}$ be a set of $\forall\exists$-rules. There is a homomorphism from F' to $\alpha_\infty(F, \mathcal{R})$ if and only if there is a homomorphism from F' to $\alpha^c_\infty(F, \mathcal{R})$.*

Proof: F' can be mapped to $\alpha_\infty(F, \mathcal{R})$ if and only if, by Property 1, there is k such that F' can be mapped to $\alpha_k(F, \mathcal{R})$. Equivalently, F' can be mapped to $\mathit{core}(\alpha_k(F, \mathcal{R}))$, and since $\mathit{core}(\alpha_k(F, \mathcal{R}))$ is isomorphic to $\alpha^c_k(F, \mathcal{R})$, F' can be mapped to $\alpha^c_k(F, \mathcal{R})$, which holds if and only if F' can be mapped to $\alpha^c_\infty(F, \mathcal{R})$. □

The next theorem states that the saturation scheme is sound and complete in the logical fragment of facts and rules.

Theorem 2 (Saturation). *Let F and F' be two facts and $\mathcal{R}$ be a set of $\forall\exists$-rules. Then $F, \mathcal{R} \models F'$ if and only if there is a homomorphism from F' to $\alpha_\infty(F, \mathcal{R})$.*

Proof: See Appendix A.

□

2.4. ENTAILMENT *and Equivalent Problems*

A knowledge base $\mathcal{K} = (F, \mathcal{R})$ is composed of a (finite) fact F and a finite set $\mathcal{R}$ of $\forall\exists$-rules. From a logical viewpoint, a (finite) fact is a $\forall\exists$-rule with an empty hypothesis, thus $\{F\} \cup \mathcal{R}$ could be seen as a set of rules. However, the distinction between both sets is meaningful from a knowledge representation viewpoint. A formula ϕ is said to be a consequence of $\mathcal{K} = (F, \mathcal{R})$ if $\{F\} \cup \mathcal{R} \models \phi$ (short notations: $\mathcal{K} \models \phi$ or $F, \mathcal{R} \models \phi$).

The following problems are fundamental on these knowledge bases:

- FACT ENTAILMENT: given a KB $\mathcal{K}$ and a (finite) fact Q, does $\mathcal{K} \models Q$ hold true?
- $\forall\exists$-RULE ENTAILMENT: given a KB $\mathcal{K}$ and a $\forall\exists$-rule R, does $\mathcal{K} \models R$ hold true?

Property 3. FACT ENTAILMENT *and* $\forall\exists$-RULE ENTAILMENT *are polynomially equivalent.*

Proof: FACT ENTAILMENT is a specific case of $\forall\exists$-RULE ENTAILMENT. For the other direction, we rely on [11], in which the "TGD implication" problem is considered, whose logical form is the same as $\forall\exists$-RULE ENTAILMENT. □

An important task on knowledge bases is query answering. Let us focus on conjunctive queries, which are considered as the basic queries in databases and knowledge-based systems. Such a query is often written *a la* Datalog: $Q = ans(x_1, \ldots, x_k) \leftarrow B$, where B (the "body" of Q) is a fact, $x_1, \ldots, x_k$ occur in B and *ans* is a special k-ary predicate, whose arguments are used to build an answer. A relational database can be identified with a ground fact. Given a ground fact D, an answer to Q in D is a tuple of constants $(a_1, \ldots, a_k)$ such that there is a homomorphism h from B to D, with $(h(x_1), \ldots, h(x_k)) = (a_1, \ldots, a_k)$. If $k = 0$, i.e., Q is a Boolean query, the unique answer to Q is the empty tuple if there is a homomorphism from B to D, otherwise there is no answer to Q. The following basic query problems are easily shown to be equivalent to FACT ENTAILMENT:

- QUERY ANSWERING decision problem: given a KB $\mathcal{K}$ and a conjunctive query Q, is there an answer to Q in $\mathcal{K}$?
- QUERY EVALUATION decision problem: given a KB $\mathcal{K}$, a conjunctive query Q and a tuple of constants t, is t an answer to Q in $\mathcal{K}$?
- BOOLEAN QUERY ANSWERING problem: given a KB $\mathcal{K}$ and a Boolean conjunctive query Q, is $()$ an answer to Q in $\mathcal{K}$?

Moreover, several fundamental problems on TGDs in databases are equivalent to the above problems, see Section 9. From now on, we consider FACT ENTAILMENT as the representative of this family of equivalent problems and simply call it ENTAILMENT. All results obtained regarding this problem can be immediately recast in terms of the other problems. Furthermore, we will simply write "rule" instead of $\forall\exists$-rule.

2.5. On the Undecidability of ENTAILMENT

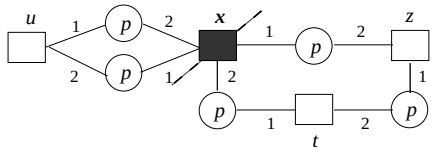
ENTAILMENT is undecidable. The oldest proofs of this result are for the equivalent problem of TGD implication [10][38][19] (see Section 9). Other proofs have been built for the entailment problem with conceptual graph rules [3] [8] (see Section 9). It is to be noticed that ENTAILMENT remains undecidable even with very strong restrictions. For instance, in the reduction of [8], the produced rules have a frontier of size 2 and the hypothesis and conclusion are paths. In [3], it is shown that ENTAILMENT can be reduced to its restriction where the vocabulary is limited to a single binary predicate. In Section 6, we show that this problem remains undecidable when the set of rules is restricted to a single rule (Theorem 8).

3. Piece-based Backward Chaining

While forward chaining uses rules to enrich facts and produce a fact to which the query maps, backward chaining proceeds in the "reverse" manner: it uses the rules to rewrite the query in different ways and produce a query that maps to the facts. The key operation in a backward chaining mechanism is the *unification* operation between part of a current goal (a conjunctive query or a fact in our framework) and a rule conclusion. This mechanism is typically used in logic programming, with rules having a single atom in the conclusion, which is unified with an atom of the current goal. Since the conclusion of a $\forall\exists$-rule has a more complex structure (it may contain several atoms and possibly existentially quantified variables), the associated unification operation is also more complex. It allows to process conclusions and goals without decomposing them into single atoms, and we will show that it is worthwhile to do so. We rely on the notion of a *piece*, which stems from a graph vision of rules and was introduced in [35] for CG rules. We reformulate it, as well as the associated unification notion, in a logical framework. As shown in Section 7.1, we will also use unification in a new perspective, namely as a tool to characterize the notion of dependency between rules.

3.1. Pieces

Given a subset T of its terms, a fact can be partitioned into pieces according to T. The piece notion is easier to grasp if we view a fact as a graph (see Section 2.2 and Figure 2). Then, given a set of term nodes T, two atom nodes a_1 and a_2 are in the same piece if there is a *path* between them that does not go *through* a node of T. If $T = \emptyset$, each connected component of T is a piece. In the following, we will impose that T contains all constant nodes in the fact, i.e., for a fact F, $T = const(F) \cup X$ with $X \subseteq var(F)$. Then, a_1 and a_2 are in the same piece if they are connected by a path of nodes corresponding to atoms and variables outside X. The next logic-based definition corresponds to this view of pieces.



$\{p(x,z), p(z,t), p(t,x), p(x,u), p(u,x)\}$ and $T = \{x\}$

Figure 2: Pieces

Definition 10 (Piece). *Let F be a fact and $X \subseteq var(F)$. A* piece *of F according to X is a minimal non-empty subset P of F such that, for all a and a' in F, if $a \in P$ and $(var(a) \cap var(a')) \not\subseteq X$, then $a' \in P$. Let $R = (H, C)$ be a rule. A piece in R is a piece of C according to $fr(R)$.*

Definition 11 (Cutpoints). *Let $R = (H, C)$ be a rule. A* cutpoint *in R is either a frontier variable or a constant in C. We note $cutp(R) = fr(R) \cup const(C)$ the set of cutpoints in R.*

The notion of a piece in R can equivalently be defined according to $cutp(R)$ instead of $fr(R)$. Indeed, only existential variables in the rule conclusion allow to "glue" atoms into pieces.

Example 1. *[Pieces] Cf. Figure 2.*
$R = p(x,y) \rightarrow p(x,z) \wedge p(z,t) \wedge p(t,x) \wedge p(x,u) \wedge p(u,x)$. The frontier of R is $\{x\}$, hence R has two pieces $\{p(x,z), p(z,t), p(t,x)\}$ and $\{p(x,u), p(u,x)\}$. Would u be replaced by a constant, the second piece would be cut into two pieces.

A piece in a rule R can be seen as a "unit" of knowledge brought by an application of R in forward chaining. Indeed, on the one hand R can be decomposed into an equivalent set of rules with the same hypothesis and exactly one piece in the conclusion:

Property 4. *Let $R = (H, C)$ be a rule and $P_1, \ldots, P_k$ be the pieces of R. Then R is equivalent to the conjunction of the rules $R_1, \ldots, R_k$, where $R_i = (H, P_i)$.*

On the other hand, the conclusions of the obtained rules cannot be further decomposed while keeping a set of $\forall\exists$-rules with the same semantics as R (provided that H is not modified, otherwise other decompositions are possible: see for instance the "atomic decomposition" in Section 3.2).

3.2. Piece-Unifiers

Backward chaining erases whole pieces of a goal Q. To explain the key ideas of the following unifier definition, let us present it as performing the inverse of a rule application to a potential fact. Given Q and a rule $R = (H, C)$, assume that Q can be proven by an application of R to a fact F according to a homomorphism π, i.e., there is a homomorphism π' from Q to $\alpha(F, R, \pi)$ and $\pi'(Q)$ is not included in F. Let σ_R be the substitution of $fr(R)$ (extracted from π) used to apply R to F. Q can then be

partitioned into Q' and Q'', such that π' maps Q' to $\sigma_R(C)$ and Q'' to F. Let T_Q be the set of terms (or simply variables) t in Q such that $\pi'(t)$ is in $\sigma_R(\mathit{cutp}(R))$.

T_Q defines pieces of Q, which can be partitioned into pieces of Q' and pieces of Q''. Each piece of Q' is mapped by π' to a piece of $\sigma_R(C)$. Roughly said, the backward chaining step associated with σ_R erases from Q the pieces composing Q', applies π' to the remaining variables of T_Q and adds $\sigma_R(H)$.

Definition 12 (Piece-unifier). *Let Q be a fact and $R = (H, C)$ be a rule. A* piece-unifier *(or simply* unifier*) of Q with R is a tuple $\mu = (T_Q, Q', \sigma_R, \pi_Q)$ where:*

- *T_Q is a (possibly empty) subset of $\mathit{var}(Q)$, which thus defines pieces in Q;*
- *Q' is the union of one or more pieces of Q according to T_Q;*
- *σ_R is a substitution of $fr(R)$ (or equivalently of $\mathit{cutp}(R)$) by $\mathit{cutp}(R) \cup \mathit{const}(Q')$;*
- *π_Q is a homomorphism from Q' to $\sigma_R(C)$ such that, for all $t \in T_Q \cap \mathit{var}(Q')$, there is $t' \in \mathit{cutp}(R)$ with $\pi_Q(t) = \sigma_R(t')$.*

Definition 13 (Rewriting of a fact). *Let Q be a fact, $R = (H, C)$ be a rule and $\mu = (T_Q, Q', \sigma_R, \pi_Q)$ be a unifier of Q with R. A* rewriting of Q according to R and μ *produces a fact $\beta(Q, R, \mu) = {\sigma_R}^{\text{safe}}(H) \cup \pi_Q(Q \setminus Q')$.*

Note that, as for the operator α that applies a rule, the operator β produces a fact that is unique up to variable renaming.

Example 2 (Piece-unifier). *Cf. Figure 3. Let $R = h(x, y) \rightarrow p(x, z) \wedge q(z, y) \wedge r(z, t)$. Let $Q = \{p(u, v), q(v, u), s(u, w)\}$. Q is unifiable with R by the following unifier $\mu = (T_Q, Q', \sigma_R, \pi_Q)$: $T_Q = \{u\}$ defines two pieces $Q_1 = \{p(u, v), q(v, u)\}$ and $Q_2 = \{s(u, w)\}$ (see Figure 3: Q_1 is colored in gray); $Q' = Q_1$; $\sigma_R = \{(y, x)\}$; $\pi_Q = \{(u, x), (v, z)\}$. The new fact $\beta(Q, R, \mu)$ is $\{h(x, x), s(x, w)\}$.*

It might be argued that the unifier notion would be simpler if rule conclusions were decomposed, not only into single pieces, but into single atoms.

Indeed, a rule (H, C) can be equivalently[2] encoded by the following set of rules: $\{(H, R(t_1, \ldots, t_k)), (R(t_1, ..., tk), A_c)_{A_c \in C}\}$, where R is a new predicate assigned to the rule and $t_1, \ldots, t_k$ are the variables in C. However, the rewriting mechanism would then build "nogood" unifications that would have been avoided with piece-based unification, as illustrated in Example 3. Besides the loss of efficiency in backward chaining, this decomposition of rule conclusions beyond single pieces weakens the characterization of decidable cases, as shown in Section 7.1.

Example 3 (Atomic decomposition). *The rule $R = h(x, y) \rightarrow p(x, z) \wedge p(z, t) \wedge p(t, x)$, which has a single piece, could be replaced by four rules: $R_1^A = h(x, y) \rightarrow R(x, z, t)$, $R_2^A = R(x, z, t) \rightarrow p(x, z)$, $R_3^A = R(x, z, t) \rightarrow p(z, t)$ and $R_4^A = R(x, z, t) \rightarrow p(t, x)$. Let $Q = p(u, v) \wedge p(v, u)$. Q is not piece-unifiable with R, but it is with R_2^A, R_3^A and R_4^A. Indeed, the information that these rules cannot be considered independently has been lost.*

[2] See Section 6.3 for a precise definition of equivalence.

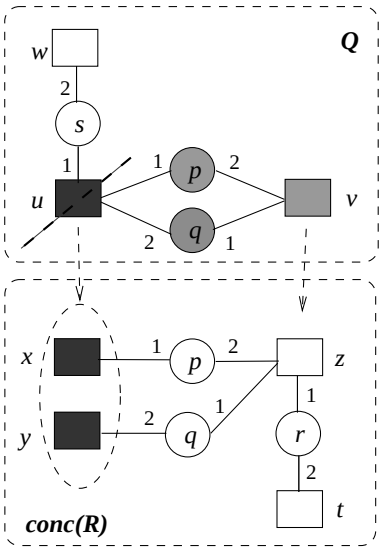


Figure 3: Piece-unifier

Moreover, having a piece restricted to an atom does not really simplify the unifier notion. The main reason of the unifier complexity is the presence of existentially quantified variables. To come to Horn rules and rely on usual binary unification, one may replace existential variables by Skolem functions (on frontier variables) and split rules into rules with an atomic conclusion. However, as for the preceding transformation, this leads to perform nogood unifications, see Example 4.

Example 4 (Transformation into Horn rules (skolemisation)). *Let us decompose the rule R in Example 3 into three Horn rules: $R_1^H = h(x,y) \rightarrow p(x,f(x))$, $R_2^H = h(x,y) \rightarrow p(f(x),g(x))$ and $R_3^H = h(x,y) \rightarrow p(g(x),x)$. $Q = \{p(u,v), p(v,u)\}$ is not piece-unifiable with R, but each atom in Q is unifiable with the conclusion of each R_i^H.*

3.3. Backward Chaining

Definition 14 (Rewriting sequence). *Let Q and Q' be two facts, and $\mathcal{R}$ be a set of rules. We say that Q' is an* $\mathcal{R}$-rewriting of *Q if there is a finite sequence (called the* rewriting sequence*) $Q = Q_0, Q_1, \ldots, Q_k = Q'$ such that for all $1 \leq i \leq k$, there is a rule $R \in \mathcal{R}$ and a unifier μ of Q_{i-1} with R such that $Q_i = \beta(Q_{i-1}, R, \mu)$.*

The soundness and completeness of backward chaining (next theorem) relies on the following equivalence between $\mathcal{R}$-rewriting and $\mathcal{R}$-derivation: given F and Q two facts and $\mathcal{R}$ a set of rules, there is an $\mathcal{R}$-rewriting of Q that maps to F if and only if there is an $\mathcal{R}$-derivation F' of F such that Q maps to F'. The precise correspondence between derivation and rewriting is stated in Lemmas 7 and 8 given in Appendix B. The proof of the following theorem relies on these lemmas.

Theorem 3. *Let $\mathcal{K} = (F, \mathcal{R})$ be a KB and Q be a fact. Then $F, \mathcal{R} \models Q$ if and only if there is an $\mathcal{R}$-rewriting of Q that maps to F.*

Proof: See Appendix B. □

4. Abstract Decidable classes

We distinguish between several kinds of known decidable cases according to the properties defining them:

- *abstract classes* are defined by abstract properties that ensure decidability but for which the existence of a procedure for deciding whether a given set of rules fullfils the property is not obvious; in fact, we will show that none of the three known abstract classes is recognizable;
- *concrete classes* are defined by syntactic properties. These properties can be defined on a set of rules–they are called *global* properties–or individually on each rule –they are called *individual* properties.

In this section, we study abstract classes. These classes will be used to structure the set of known concrete decidable classes.

Let us consider a forward chaining scheme directly based on the saturation notion. One step consists in finding all *new*[3] homomorphisms from the rule hypotheses to the current fact and performing the corresponding rule applications. Two halting conditions can be defined: if there is a homomorphism from Q to the current fact, the answer is yes; otherwise, if it is useless to continue applying the rules, the answer is no. In the second case, one can stop at a rank k because no new homomorphism is found, or because all new rule applications are redundant, which is a stronger condition and in general allows to stop sooner (for a discussion of this distinction, see [8]). Sets of rules that guarantee that such a k exists are called finite expansion sets [8]. With these sets, ENTAILMENT is obviously decidable.

Definition 15 (Finite Expansion Set). *A set of rules $\mathcal{R}$ is said to be a* finite expansion set *(*fes*) if and only if, for every fact F, there exists an integer k such that $F_k = \alpha_k(F, \mathcal{R}) \equiv F_{k+1} = \alpha_{k+1}(F, \mathcal{R})$ (i.e., all rule applications to F_k are redundant). F_k is called a* full *$\mathcal{R}$-derivation of F.*

Property 5. ENTAILMENT *is decidable if $\mathcal{R}$ is a finite expansion set of rules.*

[12] studies conditions on rules that ensure the decidability of ENTAILMENT even when the saturation is not a finite process. The following definition of an abstract class called *bounded treewidth set* of rules (Definition 17) translates the fundamental property underlying the concrete decidable classes studied in the latter paper.

[3] A homomorphism π from a rule hypothesis H to F_k is *new* if π is not a homomorphism from H to F_{k-1}.

As already mentioned, a fact can naturally be seen as a hypergraph whose nodes encode its terms and hyperarcs encode its atoms. The *primal* graph (also called Gaifman graph) of this hypergraph is the undirected graph with the same set of nodes and such that there is an edge between two nodes if they belong to the same hyperarc, i.e., the corresponding terms occur in the same atom. The following treewidth definition for a fact corresponds to the usual treewidth definition for the associated primal graph.

Definition 16 (Treewidth of a fact). *Let F be a (possibly infinite) fact. A* tree decomposition *of F is a (possibly infinite) tree $T = (\mathcal{X} = \{X_1, \ldots, X_k, \ldots\}, U)$ where:*

1. *the X_i are sets of terms of F with $\bigcup_i X_i = term(F)$;*
2. *For each atom a in F, there is $X_i \in \mathcal{X}$ such that $term(a) \subseteq X_i$;*
3. *For each term e in F, the subgraph of T induced by the nodes X_i such that $e \in X_i$ is connected.*

The width *of a tree decomposition T is the size of the largest node of T, minus 1. The* treewidth *of a fact F is the minimal width among all its possible tree decompositions.*

Definition 17 (Bounded Treewidth Set). *(basically [12]) A set of rules $\mathcal{R}$ is called a* bounded treewidth set (*bts*) *if for any fact F there exists an integer b such that, for any fact F' that can be $\mathcal{R}$-derived from F, the treewidth of $core(F')$ is less or equal to b.*

Note that, for any fact F, the treewidth of $core(F)$ is less or equal to the treewidth of F (since $core(F) \subseteq F$), thus by considering the cores of derived facts instead of the derived facts themselves, we define a larger *bts* class than the one we introduced in [5].

Theorem 4 (Decidability of bts). [4] *The restriction of* ENTAILMENT *to bounded treewidth sets of rules is decidable.*

Proof: Let $\mathcal{R} = \{R_1, \ldots, R_n\}$ be a *bts*. By definition, for any fact F, there exists a bound b such that any core (of a fact) $\mathcal{R}$-derivable from F has treewidth at most b.

Let F^* be the union of all cores of facts in the (potentially infinite) saturation of F with $\mathcal{R}$, i.e. $F^* = \alpha^c_\infty(F, \mathcal{R})$ (notation introduced below Definition 9). Thanks to the treewidth compactness theorem [37], F^* has bounded treewidth. Let F and Q be facts. By Theorems 1 and 2 , $F, \mathcal{R} \models Q$ iff $\alpha_\infty(F, \mathcal{R}) \models Q$, i.e., by Property 2, $F^* \models Q$, i.e., $F^* \wedge \neg Q$ is unsatisfiable. Let I^* be an isomorphic model of F^* (for a precise definition of isomorphic model, see Appendix A, Lemma 4). It holds that when $F^* \wedge \neg Q$ is satisfiable, then I^* is a model of it. To prove it, we use the notions introduced in Appendix A (by absurd: assume that $F^* \wedge \neg Q$ is satisfiable and I^* is not a model of it, i.e., I^* is a model of Q; then by Property 21, there is a witness of Q in I^*, hence, by Lemma 3.1, there is a homomorphism from Q to F^*, thus, by

[4] This theorem is an immediate generalization of Theorem 23 in [12], that applies to the concrete *bts* class called "weakly guarded TGD".

Theorem 1, $F^* \models Q$, which contradicts the hypothesis that $F^* \wedge \neg Q$ is satisfiable). It follows that formulas of the form $F \wedge R_1 \ldots \wedge R_n \wedge \neg Q$ have the bounded treewidth model property (i.e., they have a model of bounded treewidth, here I^*, when they are satisfiable). We conclude with [25], that states that classes of first-order logic having the bounded treewidth model property are decidable. □

Note that the proof of the previous theorem does not directly provide an algorithm for ENTAILMENT with *bts*, whereas algorithms are naturally associated with the other abstract classes (i.e., the *fes* class and the *fus* class defined hereafter).

A *fes* is a *bts*, since all full derivations of F have isomorphic cores, whose treewidth is bounded by their own size.

Backward chaining looks for a rewriting of Q that maps to F. Note that not all rewritings are useful: indeed, let Q_1 and Q_2 be two rewritings such that Q_1 maps to Q_2 (i.e., Q_1 is "more general" than Q_2); if Q_1 does not map to F, neither does Q_2.

Now, consider a backward chaining mechanism that builds $\mathcal{R}$-rewritings of Q in a breadth-first way and maintains a set $\mathcal{Q}$ of the most general $\mathcal{R}$-rewritings built, i.e., it does not add a new $\mathcal{R}$-rewriting Q'' to $\mathcal{Q}$ if there is $Q' \in \mathcal{Q}$ with $Q'' \models Q'$ (it should also remove an existing element of $\mathcal{Q}$ when a more general Q'' has been found, but this has no influence of the abstract decidable class defined); it answers yes if it finds an $\mathcal{R}$-rewriting Q' such that $F \models Q'$. This algorithm is sound and complete and halts on positive instances of the problem. Whereas finite expansion sets ensure that all information entailed by a finite fact in forward chaining can be encoded in a finite fact, the *finite unification sets* presented hereafter ensure that the above set $\mathcal{Q}$ of rewritings is finite.

Definition 18 (Finite Unification Set). *A set of rules $\mathcal{R}$ is called a finite unification set (*fus*) if for every fact Q, there is a finite set $\mathcal{Q}$ of $\mathcal{R}$-rewritings of Q such that, for any $\mathcal{R}$-rewriting Q' of Q, there is an $\mathcal{R}$-rewriting Q'' in $\mathcal{Q}$ that maps to Q'. We say that $\mathcal{Q}$ is a full $\mathcal{R}$-rewriting set of Q.*

Note that it may be the case that the set of the most general rewritings is finite while the set of rewritings is infinite. F.i. let $Q = t_1(x)$, $R_1 = p(x, y) \wedge t_2(x) \rightarrow t_1(y)$ and $R_2 = p(x, y) \wedge t_1(x) \rightarrow t_2(y)$; the set of the most general rewritings is $\{t_1(x), p(x, y) \wedge t_2(x)\}$.

The above backward chaining mechanism halts in finite time if $\mathcal{R}$ is a *fus*, hence:

Property 6. ENTAILMENT *is decidable if $\mathcal{R}$ is a finite unification set of rules.*

We show now that *fes*, *bts* and *fus* yield abstract characterizations that are not recognizable, with a proof applying to the three abstract classes. Note that the undecidability of *fes* recognition has also been proven in [27], with a reduction from the halting problem of a Turing Machine.

Theorem 5. *Deciding if a set $\mathcal{R}$ is a finite expansion (resp. finite unification, resp. bounded treewidth) set is undecidable.*

The proof of this theorem relies on the following lemmas.

Lemma 1. *Let $(F, \mathcal{R}, Q)$ be an instance of* ENTAILMENT. *Let $\mathcal{V}$ be the vocabulary obtained by considering predicates and constants occurring in F, $\mathcal{R}$ and Q. We note $\mathcal{R}' = \mathit{allrules}(F, \mathcal{R}, Q) = \mathcal{R} \cup \{(\emptyset, F); (Q, U)\}$ a new set of rules, where U is the* all-true *fact. Then $F, \mathcal{R} \models Q$ if and only if $\emptyset, \mathcal{R}' \models U$.*

Proof: Since the fact F and the rule $(\emptyset, F)$ are equivalent, we prove that $F, \mathcal{R} \models Q$ iff $F, \mathcal{R} \cup \{(Q, U)\} \models U$.
($\Rightarrow$) Immediate, since $F, \mathcal{R} \models Q$ and $Q, (Q, U) \models U$ implies $F, \mathcal{R}' \models U$.
($\Leftarrow$) If $F, \mathcal{R}' \models U$, then there exists a derivation $F = F_0, F_1, ..., F_k$ such that $F_k \models U$. Suppose that the rule $R_U = (Q, U)$ is not used in this derivation. Then $F, \mathcal{R} \models U$ and since any fact is entailed by U, we have $F, \mathcal{R} \models Q$. Otherwise, let us consider the smallest i such that F_i is obtained from F_{i-1} by an application of R_U. It means that there is a homomorphism from Q to F_{i-1} (applicability of the rule) and that $F, \mathcal{R} \models F_{i-1}$ (R_U was not needed). Then $F, \mathcal{R} \models Q$. □

Lemma 2. *Let $(F, \mathcal{R}, Q)$ be an instance of* ENTAILMENT. *Let $\mathcal{R}' = \mathit{allrules}(F, \mathcal{R}, Q)$ be defined as in Lemma 1. If $\emptyset, \mathcal{R}' \models U$, then $\mathcal{R}'$ is a* fes*, a* fus *and a* bts*.*

Proof: Assume $\emptyset, \mathcal{R}' \models U$. We successively prove three implications:
1) It follows that, for any fact H on $\mathcal{V}$, we have $H, \mathcal{R}' \models U$ and then the forward chaining algorithm produces in finite time a fact F' such that $F' \models U$ (from semi-decidability of ENTAILMENT proven with saturation, see Property 1). Thus $F' \equiv U$ and any fact that can be derived from F' is also equivalent to U: it means that $\mathcal{R}'$ is a *fes*.
2) Since all *fes* are also *bts*, $\mathcal{R}'$ is also a *bts*.
3) It follows that, for any fact Q', we have $\emptyset, \mathcal{R}' \models Q'$ and then a breadth-first exploration of all possible rewritings of Q' will produce $\emptyset$ in finite time (from semi-decidability of ENTAILMENT proven with backward chaining). Since $\emptyset$ is more general than any other rewriting of Q', $\mathcal{R}'$ is a *fus*. □
Proof: [Theorem 5] (By absurd). Assume there exists a halting, sound and complete algorithm that determines whether a set of rules is a *fes* (resp. a *bts*, resp. a *fus*). Then we exhibit the following halting, sound and complete algorithm for ENTAILMENT.

Data: $(F, \mathcal{R}, Q)$ an instance of ENTAILMENT
Result: YES iff $F, \mathcal{R} \models Q$, NO otherwise.
1 **if** $\mathcal{R}' = \mathit{allrules}(F, \mathcal{R}, Q)$ *is a* fes *(resp.* fus*, resp.* bts*)* **then**
2 | **return** YES *iff* $\emptyset, \mathcal{R}' \models U$*, and* NO *otherwise*;
3 **else return** NO;

This algorithm halts: the condition in line 1 is checked in finite time (by hypothesis), and if this condition is fulfilled then the consequence (line 2) can also be checked in finite time. This algorithm is sound and complete: line 2 returns the correct answer (Lemma 1) and, assuming that the condition is not verified, line 3 also returns the correct answer (from Lemma 1 and contrapositive of Lemma 2, we have "if $\mathcal{R}'$ is not a *fes* then $F, \mathcal{R} \not\models Q$"). □

5. Concrete Decidable Classes

In this section, we first review known concrete classes implementing the *fes* or *bts* behaviors, and exhibit a generalization of them, which is still *bts*. Then, we introduce concrete classes implementing the *fus* behavior. We end with a synthetic map of all inclusions between decidable classes.

5.1. Known fes *or* bts *Concrete Classes*

A well-known concrete case of *fes* is that of *range-restricted rules* [1][5], whose conclusion does not introduce new variables: a rule $R = (H, C)$ is said to be range-restricted (*rr*) if *var*$(C) \subseteq$ *var*(H). *rr*-rules are exactly the rules in positive Datalog and more generally they are widely used in logic programming. They typically allow to express specialization relationships and properties of relations in ontological languages, such as reflexivity, symmetry or transitivity. Another concrete case of *fes* is that of *disconnected rules* (*disc*), whose frontier is empty [8]. Note that the hypothesis and the conclusion of a *disc*-rule may share constants, which allows to express knowledge about specific individuals. The reason why a set of *disc*-rules is a *fes* is that a disconnected rule needs to be applied only once: any further application of it is redundant.

Let us now consider known concrete *bts* but not *fes* classes. Two classes are based on individual criteria: rules with a *frontier of cardinality exactly one (fr1)*, mentioned in [7]; and *guarded* rules (*g*), in which an atom in the hypothesis contains ("guards") all variables of the hypothesis, studied in [12]. A subclass of guarded rules are the so-called *inclusion dependencies* (*ID*) in databases: these rules have exactly one atom in the hypothesis and in the conclusion.

Note that *fr1*-rules, *g*-rules and *disc*-rules are incomparable classes. However, they all prevent the creation of cycles with unbounded length by controlling the way knowledge added by a rule conclusion is "connected" to the current fact. This property is made explicit by the class of *frontier-guarded rules* introduced hereafter, which generalizes them: the crucial point is to guard the frontier of rules. We will prove that frontier-guarded rules, hence *g*-rules and *fr1*-rules, form a *bts* class in Section 5.2.

Example 5 (fes/bts concrete classes with individual properties).
The following rules show that rr*,* disc*,* fr1 *and* g *are pairwise incomparable classes.*
$R_1 = r(x, y) \wedge r(y, z) \rightarrow r(x, z)$ *is only* rr*;*
$R_2 = r(x, y) \wedge r(y, z) \rightarrow r(u, v)$ *is only* disc*;*
$R_3 = r(x, y) \wedge r(y, z) \rightarrow r(z, u)$ *is only* fr1*;*
$R_4 = r(x, y) \wedge r(y, z) \wedge t(x, y, z) \rightarrow t(y, z, u)$ *is only* g.

Let us now turn our attention to concrete classes defined by global properties. The *g*-rule class is generalized by the class of *weakly guarded rules* (*wg*), in which only some variables of the hypothesis need to be guarded [12]. Given a set of rules $\mathcal{R}$, a

[5] These rules are also called *full* implicational dependencies [19] and *total* tuple-generating dependencies [11] in databases.

position i in a predicate p (notation (p, i)) is said to be *affected* if it may contain a new variable generated by forward chaining. More precisely, the set of affected positions w.r.t. $\mathcal{R}$ is the smallest set that satisfies the following conditions: (1) if there is a rule conclusion containing an atom with predicate p and an existentially quantified variable in position i, then position (p, i) is affected; (2) if a rule hypothesis contains a variable x appearing in affected positions only and x appears in the conclusion of this rule in position (q, j) then (q, j) is affected. Given $\mathcal{R}$, a weak guard in a rule $(H, C) \in \mathcal{R}$ is an atom in H that guards all variables in H that occur only in affected positions; these variables are said to be *affected*. $\mathcal{R}$ is said to be *weakly guarded* if each rule in $\mathcal{R}$ has a weak guard. *wg* are shown to be *bts* in [12]. Special cases of *wg*-rules are *g*-rules (a guard is a weak guard) and *rr*-rules (no position is affected), both based on individual properties. In Section 5.2, we will generalize weakly guarded rules into weakly frontier-guarded rules and prove that we still have a *bts* class.

Two other concrete classes defined by global properties found in the literature are *weakly acyclic rules* (wa) [29][28] and sets of rules with an *acyclic graph of rule dependencies* ($aGRD$) [4]. Both are *fes* classes. The first class relies on a graph, introduced for TGD and called *dependency graph*[6], which encodes variable sharing between positions in predicates. The nodes represent the positions in predicates (cf. the notation *(p,i)* introduced for *wg* rules). For each rule $R = (H, C)$ and each variable x in H occurring in position (p, i): if $x \in fr(R)$, there is an arc from (p, i) to each position of x in C; furthermore, for each existential variable y in C (i.e., $y \in var(C) \setminus fr(R)$) occurring in position (q, j), there is a special arc from (p, i) to (q, j). The set of rules is weakly acyclic if its dependency graph has no circuit passing through a special arc. The second class relies on another graph, called the *graph of rule dependencies*, which encodes possible interactions between rules: the nodes represent the rules and there is an arc from R_i to R_j if an application of the rule R_i may create a new application of the rule R_j (with this abstract condition being effectively implemented by a unification operation, see Section 7). $aGRD$ is the case where this graph is without circuit[7].

Example 6 (bts concrete classes with global properties).
On Example 5: $\{R_2\}$ *is not* wg *because both positions in* r *are affected, thus* x, y *and* z *are affected but no atom guards them all. The same holds for* $\{R_3\}$*;*
$S_1 = \{q(x) \rightarrow p(z, x),\ p(x, z) \wedge p(y, z) \rightarrow r(x, y)\}$ *is* wa *and* aGRD *but it is not* wg *because the variables* x *and* y *in the second rule are affected but not guarded;*
$S_2 = \{p(x, y) \rightarrow p(y, z)\}$ *is* wg *(because the rule is* g*) but it is not* wa *neither* aGRD *(this rule depends on itself);*
$S_3 = \{p(x, y) \wedge q(y) \rightarrow p(y, z) \wedge s(z)\}$ *is* aGRD *and* wg *(because the rule is* g*) but it is not* wa*;*
$S_4 = \{q(x) \wedge p(x, y) \rightarrow q(y) \wedge r(y, z)\}$ *is* wa *and* wg *(because the rule is* g*) but it is not* aGRD *(this rule depends on itself).*

[6] We use here the terminology of [29], developed in [30].

[7] Unfortunately, the term "acyclic" is ambiguous when used on directed graphs. In this paper, by acyclic we mean without any undirected cycle (i.e., the underlying undirected graph is a forest). We keep the expressions "acyclic GRD" and "weakly acyclic" that come from other papers, but precise that they refer to circuits.

Note that these sets show that wg*,* wa *and* aGRD *are pairwise incomparable.*

5.2. Generalizations of bts *Concrete Classes*

Definition 19 ((Weakly) frontier-guarded rules). *Given a set of variables S, a rule is* S-guarded *if an atom of its hypothesis contains (at least) all variables in S. A rule is* frontier-guarded *(*fg*) if it is S-guarded with S being its frontier. A set of rules is* weakly frontier-guarded *(*wfg*) if each rule is S-guarded with S being the set of affected variables in its frontier.*

The class of frontier-guarded rules includes *g*-rules, *fr1*-rules and *disc*-rules. The class of weakly frontier-guarded rules generalizes it as well as the class of weakly guarded rules, which itself generalizes range-restricted rules. In particular, it covers all known concrete decidable classes (to the best of our knowledge) having the *bounded treewidth set* property and based on individual criteria.

Example 7 (bts concrete classes (continued)).
$R_5 = r(x, y) \wedge r(y, z) \rightarrow s(x, u) \wedge s(y, u)$ *is not* g *nor* fr1 *but it is* fg*;*
$R_6 = r(x, w) \wedge s(y, z) \rightarrow r(u, x) \wedge s(y, u)$ *is not* fg*;* $\{R_6\}$ *is not* wg *either (the affected variables in H are x, w and z), but it is* wfg *(since $r(x, w)$ guards x, which is the only variable both affected and in the frontier).*

Rules allow us to express some description logic statements, especially those forming the kernel of recent DLs directed towards efficient query answering, typically: inclusions between concepts built with conjunction ($\sqcap$) and full existential restriction ($\exists r.C$), as well as role inclusions, domain and range restrictions, reflexivity and transitivity role properties etc... The first-order translation of these statements yields rules that, besides the fact that they have an "acyclic" hypothesis and conclusion, are special cases of previous concrete classes. For instance, [13] shows that the major members of the DL-Lite family [18] are covered by guarded rules (plus specific equality rules and negative constraints, which do not interfere with query answering, see Section 9). Another example of DL covered by a decidable concrete class of $\forall\exists$-rules is $\mathcal{ELH}_{\bot}^{dr}$ [33] (which can be seen as the core of the $\mathcal{EL}$ profile in the Semantic Web ontological language OWL2): it can be easily checked that all inclusions[8] in this DL are *fr1*-rules or *ID*, thus they are *fg*-rules. E.g., the following $\mathcal{ELH}_{\bot}^{dr}$ inclusion: $\exists r.C \sqcap \exists r.D \sqsubseteq \exists s.E$ can be translated into the rule $r(x, y) \wedge C(y) \wedge r(x, z) \wedge D(z) \rightarrow s(x, u) \wedge E(u)$, which is *fr1*, hence *fg*. Rules expressing transitivity are not *fg*, but *rr*, thus both *wg* and *wfg*. The weakly frontier-guarded class thus seems particularly appropriate for studying these new DLs as rules.

In the following, we prove that (weakly) frontier-guarded rule sets are bounded treewidth sets. For that, we introduce the notion of a derivation graph. This graph is of interest in itself because it allows to explain properties of rules by structural properties of the facts they produce. We call *frontier atom* in a rule R an atom in the hypothesis of R that contains at least one frontier variable. Frontier atoms play an important role in the next definitions.

[8] We assume that $\bot$ is processed by a negative constraint, which does not interfere with query answering.

Definition 20 (Derivation Graph). *Let $D = (F = F_0, F_1, \ldots, F_n = F')$ be a derivation sequence. The Derivation Graph assigned to D is the directed graph $G_D = (\mathcal{X}, E, newAtoms, label)$, where $\mathcal{X}$ is the set of nodes, E is the set of arcs, and newAtoms and label are functions respectively labeling nodes and arcs, such that:*

- $\mathcal{X} = \{X_0, \ldots, X_n\}$;
- *newAtoms assigns to each $X_i \in \mathcal{X}$ the set of atoms created at step i, i.e., $newAtoms(X_0) = F$ and for $1 \leq i \leq n, newAtoms(X_i) = F_i \setminus F_{i-1}$. Furthermore, we note $term(X_i) = term(newAtoms(X_i))$ (and we will often write "X_i contains t" instead of "$t \in term(X_i)$");*
- *there is an arc (X_i, X_j) in E if: let $F_j = \alpha(F_{j-1}, R, \pi)$; there are $a \in newAtoms(X_i)$ and b a frontier atom in R with $\pi(b) = a$; $label(X_i, X_j) = \{e \in term(X_i) | \exists a \in newAtoms(X_i)$ such that $e \in term(a)$, $\exists b$ frontier atom in R with $x \in term(b) \cap fr(R)$, $\pi(b) = a$ and $\pi(x) = e\}$.*

Roughly speaking, nodes and their labeling encode atoms created at each derivation step; each arc (X_i, X_j) expresses that the homomorphism π from a rule hypothesis H to F_{j-1}, that has led to F_j, has mapped at least one frontier atom in H to an atom (in F_{j-1}) created in F_i; the label of (X_i, X_j) indicates the terms in F_i that are used to produce the new atoms in F_j. By definition, a derivation graph has no circuit, but it is generally not acyclic (i.e., it is not a tree, or a forest if not connected). Every application of a disconnected rule leads to a node initially isolated, thus the graph may be not connected.

Example 8 (Derivation graph). *Let $\mathcal{R}$ be the set of rules composed of the four following rules:*

$$R_1 = q(x,y) \rightarrow p(y,z), \quad R_2 = p(x,y) \wedge p(y,z) \rightarrow q(y,w) \wedge q(z,w) \wedge p(w,C),$$

$$R_3 = p(x,y) \wedge q(y,z) \rightarrow q(x,A) \wedge p(A,z), \quad R_4 = q(x,x) \wedge p(x,y) \rightarrow q(y,y)$$

and let $F = \{q(x_1,x_1), q(x_1,C)\}$ be a fact. Figure 4 shows the derivation graph associated with the derivation sequence $F = F_0,\ F_1,\ F_2,\ F_3,\ F_4,\ F_5 = F'$, where each F_i ($0 < i \leq 5$) is obtained from F_{i-1} by the following rule applications:

$$F_1 = \alpha(F_0, R_1, \{(x,x_1),(y,x_1)\}) = F_0 \cup \{p(x_1,x_2)\}$$

$$F_2 = \alpha(F_1, R_4, \{(x,x_1),(y,x_2)\}) = F_1 \cup \{q(x_2,x_2)\}$$

$$F_3 = \alpha(F_2, R_1, \{(x,x_2),(y,x_2)\}) = F_2 \cup \{p(x_2,x_3)\}$$

$$F_4 = \alpha(F_3, R_2, \{(x,x_1),(y,x_2),(z,x_3)\}) = F_3 \cup \{q(x_2,x_4), q(x_3,x_4), p(x_4,C)\}$$

$$F_5 = \alpha(F_4, R_3, \{(x,x_1),(y,x_2),(z,x_2)\}) = F_4 \cup \{q(x_1,A), p(A,x_2)\}$$

Each node X_i is labeled with $newAtoms(X_i)$. An arc (X_i, X_j) indicates that the rule application leading to X_j has mapped a frontier atom to an atom in X_i. For instance, the application of R_3 leading to X_5 has mapped the frontier atoms $p(x,y)$

(with frontier variable x) and $q(y, z)$ (with frontier variable z) respectively to $p(x_1, x_2)$ in X_1 and $q(x_2, x_2)$ in X_2, hence the arcs (X_1, X_5) and (X_2, X_5). Since x is mapped to x_1, (X_1, X_5) is labeled by x_1. Since z is mapped to x_2, (X_2, X_5) is labeled by x_2. Note that x_2 is shared by the labels of X_1 and X_5 but it is not the image of a frontier variable when $p(x, y)$ is mapped, thus it is not considered as coming from X_1.

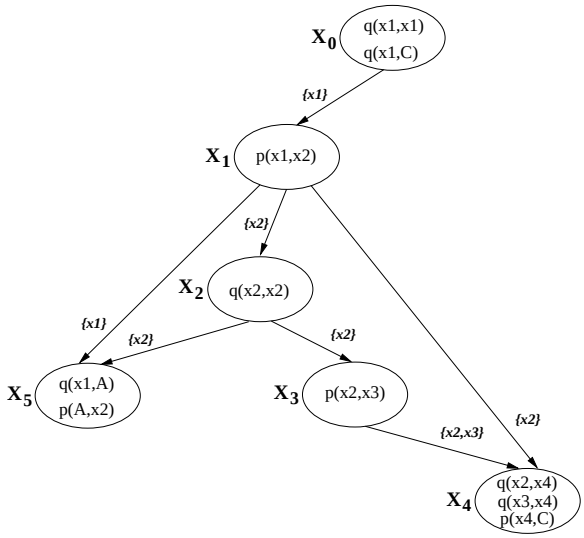


Figure 4: Derivation graph of Example 8

Property 7 (Decomposition properties). *Let $(F, \mathcal{R})$ be a KB such that no rule in $\mathcal{R}$ has a constant in its conclusion. Then, for any $\mathcal{R}$-derivation D from $F = F_0$ to $F_n = F'$, G_D satisfies the following properties, called the* decomposition properties w.r.t. F':

1. $\bigcup_i$ *term*(X_i) = *term*(F')*;*
2. *For each atom a in F', there is $X_i \in \mathcal{X}$ such that $a \in$ newAtoms(X_i);*
3. *For each term e in F', the subgraph of G_D induced by the nodes X_i such that $e \in$ term(X_i) is connected.*
4. *For each $X_i \in \mathcal{X}$, the size of term(X_i) is bounded by an integer that depends only on the size of the KB (more precisely: $max(|term(F)|, |term(C_i)|_{R_i \in \mathcal{R}})$.*

Proof: The proof of conditions 1), 2) and 4) being immediate, we focus on condition 3). Every arc labeled e (i.e., such that e belongs to its label) links two nodes containing e. For each term e in F', there exists X_e a node corresponding to F_e, the first derived fact in which e appears (if e has been generated by a rule application then X_e identifies that rule application, otherwise e belongs to F and $X_e = X_0$). Moreover, if X_i contains a

term e then F_e (the fact associated to X_e) has been generated before F_i in the derivation sequence. We can thus establish the following property: "for each node X_i such that $e \in term(X_i)$, there exists a path from X_e to X_i in which all nodes contain e and all arc labels contain e", which can be proven by induction on the length of the derivation from F_e to F_i. □

Note that the third decomposition property is not true for constants occurring in a rule conclusion. We will process these constants in a special way together with the notion of affected variable.

Property 7 expresses that D_G satisfies the properties of a tree decomposition of F' (seen as a graph) except that it is not —yet— acyclic. We now introduce operations that allow to build an acyclic graph from D_G for some classes of rules, while keeping these properties.

Definition 21 (Reduction operations on derivation graphs).

- ***Redundant arc removal.*** *Let (X_i, X_k) and (X_j, X_k) be two arcs with the same endpoint. If a term e appears in label$((X_i, X_k))$ and label$((X_j, X_k))$, then e can be removed from one of the label sets. If the label of an arc becomes empty, then the arc is removed.*
- ***Arc contraction.*** *Let (X_i, X_j) be an arc. If term$(X_j) \subseteq$ term(X_i) then X_i and X_j can be merged into a node X such that newAtoms$(X) =$ newAtoms$(X_i) \cup$ newAtoms(X_j). This merging involves the removal of (X_i, X_j) and, in all other arcs incidental to X_i or X_j, X_i and X_j are replaced by X, with multiarcs being replaced by a single arc labeled by the union of their labels.*

Example 8 (Continued). *In the derivation graph of Figure 8, one can remove the arc (X_1, X_4), which is redundant with the arc (X_3, X_4), and contract the arc (X_1, X_2).*

Property 8. *The above operations preserve the decomposition properties w.r.t. F'.*

Proof: Conditions 1), 2) and 4) are trivially respected by both operations. No atom (and thus no term) disappears in the derivation graph and no node receives any additional atom (since the only merging of nodes happens when a set is included in the other).

Condition 3) is satisfied by arc contraction, which does not change the connectivity of the graph. Let us consider redundant arc removal. For each node X that contains a term e there exists a path from X_e to X (see proof of Property 7) in which all nodes and arcs are labeled e. Moreover, the extremities of an arc labeled e also contain e, thus if (X_i, X_k) and (X_j, X_k) are labeled e, both X_i and X_j contain e, hence there is a second path from X_e to X_k. By removing one of these arcs, one does not disconnect the set of nodes containing e. □

Theorem 6. *Let $\mathcal{R}$ be a set of rules without constant in the conclusion. If for all $\mathcal{R}$-derivation D, G_D can be reduced to an acyclic graph then $\mathcal{R}$ is a bounded treewidth set.*

Proof: Follows from Property 7 and 8. □

Property 9. *If all rules are range-restricted and without constant in their conclusion, then any derivation graph with these rules can be reduced to a single node by a sequence of arc contractions.*

Proof: If all rules are *rr*, all terms in generated atoms are contained in the root of the derivation graph. We can thus iteratively contract all arcs of the derivation graph into the root. □

Property 10. *If all rules are frontier-guarded and without constant in their conclusion, then any derivation graph with these rules can be reduced to an acyclic graph.*

Proof: We show that if a node X of the derivation graph is the destination of $n \geq 2$ distinct arcs, then $n - 1$ of them can be suppressed by redundant arc removal. We begin by pointing out that, to be the destination of an arc, X must have been obtained by applying some rule R that contains at least one frontier variable (i.e., R is not *disc*). Moreover, by definition of a derivation graph, these arc labels are necessarily a subset of the terms that were images of the frontier of R. In frontier-guarded rules, a guard g of R (i.e., one of the atoms containing the frontier) generates an arc (X_g, X) in the derivation graph, where X_g contains the image of g. This arc is labeled by all terms of the frontier of R, thus the label of any other arc (X_i, X) is included in that of (X_g, X) and (X_i, X) can be removed. □

Property 11. *Frontier-guarded rules without constant in their conclusion are* bts.

Proof: Immediate consequence of Property 10 and Theorem 6. □

To cover rules that introduce constants, as well as weakly frontier-guarded rules, we extend the notion of derivation graph.

Definition 22 (Extended derivation graph). *Given a set of terms T and a derivation graph G_D, the extension of G_D with T, notation $G_D[T]$, is obtained from G_D with the following sequence of operations:*

1. *the mapping term is modified: for each X_i, term(X_i) = term(newAtoms$(X_i)) \cup T$ (i.e., the terms of T are added everywhere);*
2. *all terms occurring in T are removed from the labels in arcs; if a label becomes empty, then the arc is removed;*
3. *for each connected component (of the obtained graph) that does not include X_0, a node X_i without incoming arc is chosen and the arc (X_0, X_i) is added with label T.*

Example 8 (Final). *Figure 5 presents the extended derivation graph with $T = \{A, C\}$ (the constants occurring in F and $\mathcal{R}$) of the derivation graph given in Figure 8 after performing the two above reduction operations. The right part of each X_i shows term(X_i). The obtained graph is acyclic and satisfies the decomposition properties. The maximal number of terms in an X_i is 5, thus the treewidth of F' is less or equal to 5.*

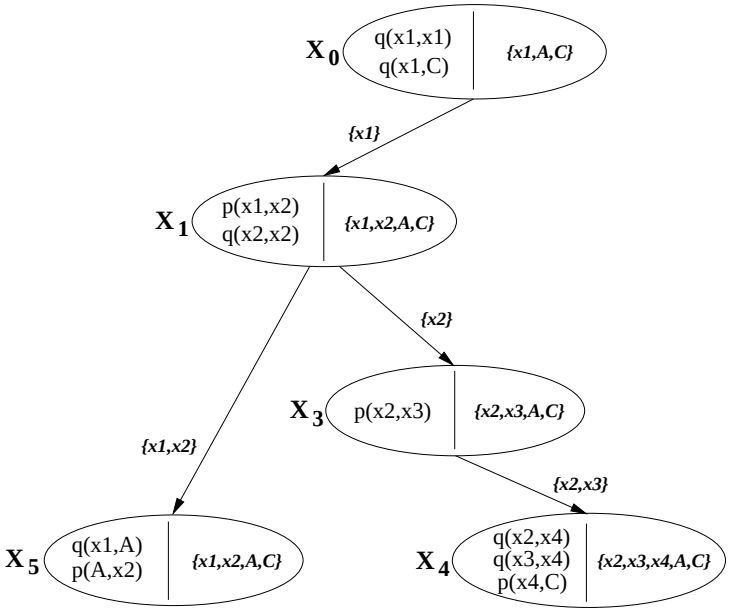


Figure 5: Extended derivation graph of Example 8

Property 12. *$G_D[T]$ satisfies the decomposition properties, with the bound on $|term(X_i)|$ being increased by $|T|$; furthermore $G_D[T]$ does not contain new cycles w.r.t. G_D.*

Proof: There is no suppression of atoms so the terms and atoms of F' remain covered. We add at most $|T|$ terms to each node thus the width of the decomposition associated with the derivation graph (and consequently the treewidth of the derived fact) is at most increased by $|T|$. Global connectivity is ensured since T is added to all nodes of the derivation graph. Since arcs are added only to reconnect disconnected components, no circuit is created. □

Theorem 6 and Property 9, 10, 11 can be extended to rules with constants in their conclusion by considering the extended derivation graph with T being the set of constants occurring in rule conclusions. In particular, this allows to ensure the third decomposition property. Property 10 is extended as follows:

Property 13. *Let $\mathcal{R}$ be a set of rules and let C be the set of constants occurring in the rule conclusions. If $\mathcal{R}$ is*frontier-guarded*, then, for any $\mathcal{R}$-derivation D, $G_D[C]$ can be reduced to a tree.*

To extend the previous property to weakly frontier-guarded rules, we add the terms of the initial fact F to T. Indeed, a non-affected variable in a rule hypothesis is necessarily mapped to a term of F by an application of this rule.

Property 14. *Let $\mathcal{R}$ be a set of rules and let C be the set of constants occurring in the rule conclusions. If $\mathcal{R}$ is* weakly frontier-guarded*, then, for any $\mathcal{R}$-derivation D, $G_D[C \cup term(F)]$ can be reduced to a tree.*

Proof: The proof is similar to the proof of Property 10. We show that if a node X in $G_D[C \cup \mathit{term}(F)]$ is the destination of $n \geq 2$ distinct arcs, then $n-1$ of them can be suppressed by redundant arc removal. X must have been obtained by applying a rule R that contains at least one frontier variable (i.e., R is not *disc*). By definition of a derivation graph, the labels of these arcs to X are necessarily subsets of the terms that were images of the frontier variables in R; furthermore, by definition of $G_D[C \cup \mathit{term}(F)]$, they do not contain terms in $C \cup \mathit{term}(F)$. If R is weakly-guarded, a weak guard g of R (i.e., one of the atoms containing the affected variables in the frontier) generates an arc (X_g, X) in the derivation graph, where X_g contains the image of g. This arc is labeled by all terms that were images of the affected variables in the frontier of R. Since non-affected variables in a rule hypothesis are necessarily mapped to a term in F by an application of this rule, the arc (X_g, X) is labeled by all terms that were images of the frontier of R (except for terms in $C \cup \mathit{term}(F)$). Thus the label of any other arc (X_i, X) is included in that of (X_g, X) and (X_i, X) can be removed. □

Theorem 7. *Weakly frontier-guarded rule sets are* bts.

Proof: Follows from Property 14 and the extension of Theorem 6. □

5.3. Fus *Concrete Classes*

We provide two concrete cases of *fus* rules hereafter (first introduced in [6]).

Definition 23 (Atomic-hypothesis rule). *A rule $R = (H, C)$ is called an* atomic-hypothesis rule *(*ah*) if H contains a single atom.*

Since ah-rules are special guarded rules, they form a decidable class. They are also a concrete case of *fus*, which yields another and simple decidability proof.

Property 15. *A set of* ah*-rules is a* fus.

Proof: Let Q be any fact. Let $R = (H, C)$ be an *ah*-rule, and $\mu = (T_Q, Q', \sigma_Q, \sigma_C)$ be a unifier of Q with R. We have $|Q| \geq |\beta(Q, R, \mu)|$ (since the rewriting removes Q', which is non-empty, and adds a specialization of the unique atom in H). Up to isomorphism, there is a bounded N number of facts of size (i.e., number of atoms) less or equal to $|Q|$ built from the bounded number of facts of constants and predicates appearing in the KB. Thus if $\mathcal{R}$ contains only *ah*-rules, the number of $\mathcal{R}$-rewritings of Q is bounded by N. □

Atomic-hypothesis rules are particularly well adapted to express necessary properties of concepts or relations in ontological languages, without any restriction on the form of the conclusion i.e., rules of form $C(x) \rightarrow P$ or $r(x_1, \ldots, x_k) \rightarrow P$, where C is a concept, r a k-ary relation and P any set of atoms.

The second kind of rules does not put any restriction on the form of the hypothesis but constrains the form of the conclusion:

Definition 24 (Domain-restricted rule). *A rule $R = (H, C)$ is called a domain-restricted rule (*dr*) if each atom of C contains all or none of the variables in H.*

Property 16. *A set of* dr *rules is a* fus.

Proof: Let us call k-limited fact, a fact Q such that each piece P of Q according to *const*(Q) fulfills $|var(P)| \leq k$. There is a finite number N of facts with at most k variables (up to isomorphism), and a bounded number of constants and predicates. Thus, a k-limited fact containing more than N pieces contains equivalent pieces, which can be removed to obtain a fact with at most N pieces. We observe that if a rewriting according to a *dr*-rule produces a new variable, then it also produces a new piece which is the only piece containing this variable. Indeed, either the unifier concerns an atom that contains all variables of H and it does not generate new variables, or it creates a new piece. Thus, if $\mathcal{R}$ contains only *dr*-rules, the number of $\mathcal{R}$-rewritings of Q without duplicate pieces is bounded (a rough upper bound is 2^N). □

E.g., the rule $R = \textit{Human}(x) \rightarrow \textit{Parent}(y, x) \wedge \textit{Human}(y)$ is both *ah* and *dr*.

5.4. Synthetic map

It can be immediately checked that *fes/bts* and *fus* are incomparable sets w.r.t. to inclusion. E.g., the set $\{R\}$ with $R = \textit{Ancestor}(x, y) \wedge \textit{Ancestor}(y, z) \rightarrow \textit{Ancestor}(x, z)$ is a *fes* (R is *rr*) but it is not a *fus*. The set $\{t(x) \rightarrow s(x, y) \wedge t(y)\ ;\ t(x) \wedge t(y) \rightarrow r(x, y)\}$ is a *fus* (it is a set of *dr* rules), but it is not a *bts* (it generates an infinite fact that is not redundant and contains a complete graph). Figure 6 synthesizes inclusions between decidable cases[9]. All inclusions are strict and no inclusion is omitted (i.e., classes not related in the schema are indeed incomparable). The preceding examples allow to check most cases and it is easy to build other examples for the missing cases.

6. Study of the Union of Decidable Classes

Before studying the decidability of the union of decidable cases, we prove a preliminary result: a single rule can make ENTAILMENT undecidable.

6.1. Undecidability with a Single rule

Theorem 8. ENTAILMENT *remains undecidable when the set of rules is restricted to a single rule.*

Proof: Let $I = (F, \mathcal{R}, Q)$ be an instance of ENTAILMENT. By a transformation τ, we build another instance $\tau(I) = (\tau(F), \tau(\mathcal{R}), \tau(Q))$ with $|\tau(\mathcal{R})| = 1$, such that I is a positive instance if and only if $\tau(I)$ is. τ is defined as follows:

- Let $\mathcal{V}$ be the vocabulary composed of the constants and the predicates occurring in I. We consider a vocabulary $\mathcal{V}_\tau$ obtained from $\mathcal{V}$ by replacing each predicate of arity k by a predicate (of same name) of arity $k + 1$ and by adding two new constants f and g (f for "fact" and g for "garbage"). Given any fact F' on $\mathcal{V}$, we denote by $\tau(F', t)$ the translation that translates each atom $p(t_1, \ldots, t_k) \in F'$ into $p(t_1, \ldots, t_k, t)$. In the following, t is either f (stating that this atom corresponds to an atom in F or entailed by F), or g.

[9] Let us mention that another concrete *fus* class has been exhibited very recently: sticky rules [15], which are incomparable with *ah*-rules and *dr*-rules.

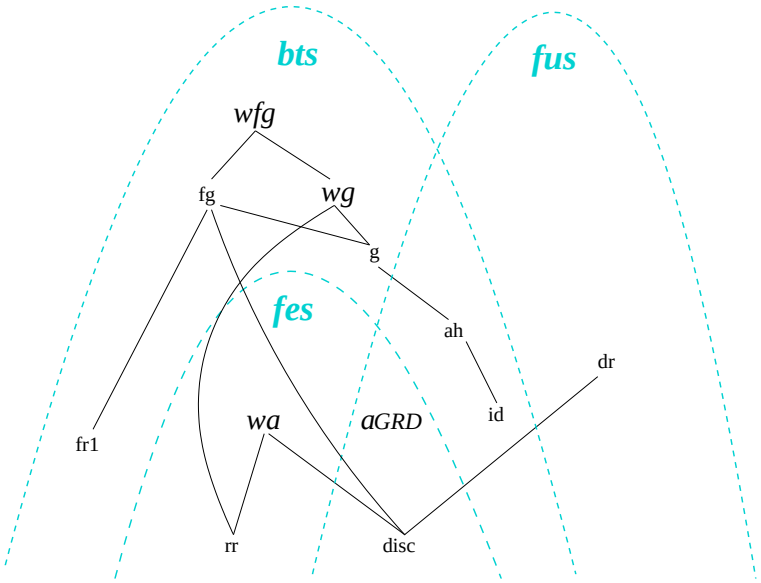


Figure 6: Inclusions between decidable cases

- $\tau(F)$ is the disjoint union of two sets: a set $C_f = \tau(F, f)$ (the "fact component") and a set $C_g = \tau(U, g)$ (the "garbage component") where U is the *all-true* fact on $\mathcal{V}$ (since any fact on $\mathcal{V}$ is entailed by U, this latter fact encodes that everything is true, but it is garbage).
- $\tau(Q) = \tau(Q, f)$ (we want to obtain it from the part of $\tau(F)$ that corresponds to F).
- Let $\mathcal{R} = \{R_1, \ldots, R_p\}$. W.l.o.g., assume that the sets of variables occurring in each rule are pairwise disjoint. Let $x_1, \ldots, x_p$ be new variables, i.e., not occurring in $\mathcal{R}$. Then $\tau(R_i) = (\tau(H_i, x_i), \tau(C_i, x_i))$.

  $\tau(\mathcal{R}) = \{R = (\cup_i \tau(H_i, x_i), \cup_i \tau(C_i, x_i))\}$ is composed of a single rule that encodes all the previous ones.

Let us outline the main ideas of this transformation. Every rule in $\mathcal{R}$ is applicable to the garbage component C_g, thus R is applicable to C_g, with variables $x_1, \ldots, x_p$ being mapped to the constant g. When a rule R_i is applicable to F by a homomorphism π, then R is applicable to $\tau(F)$ with $\tau(H_i, x_i)$ being mapped to C_f by $\pi \cup \{(x_i, f)\}$, and the remaining H_j in H being mapped either to C_f or C_g. Conversely, assume that R is applicable to $\tau(F)$: each $\tau(H_i, x_i)$ is necessarily mapped to C_f or to C_g; if $\tau(H_i, x_i)$ is mapped to C_f, this corresponds to an application of R_i to F. If all $\tau(H_i, x_i)$ are mapped to C_g then the corresponding application of R is redundant (by definition of the *all-true* fact). It follows that every derivation from F with the rules in $\mathcal{R}$ can be translated into a derivation from $\tau(F)$ with R (with a natural extension

of the homomorphisms involved in the first derivation) and reciprocally (with a natural decomposition of the homomorphisms involved in the second derivation). Finally, π is a homomorphism from Q to a fact F' defined on $\mathcal{V}$ iff it is a homomorphism from $\tau(Q)$ to $\tau(F')$ (and we have $\pi(\tau(Q)) \subseteq C'_f$ with $C'_f = \tau(F', f)$).

Note that $k+1$-ary predicates are not required to obtain the result: the same result can be obtained by decomposing k-ary predicates with $k \geq 2$ into binary predicates. □

Let us say that two classes are *compatible* if the union of any two sets respectively belonging to these classes is decidable. Otherwise, they are said to be incompatible.

6.2. Universal Compatibility of Disconnected Rules

We first prove that disconnected rules are compatible with any decidable set of rules.

Theorem 9. *Let $\mathcal{R}_1 = \mathcal{R}_0 \cup \mathcal{R}_{disc}$ be a set of rules, where R_{disc} is a set of disconnected rules. If $\mathcal{R}_0$ is decidable, then $\mathcal{R}_1$ also is.*

Proof: Let us recall that a disconnected rule needs to be applied only once. Assume we have an algorithm for ENTAILMENT, say *Ded*$(F, \mathcal{R}, Q)$, that decides in finite time for $\mathcal{R} = \mathcal{R}_0$ if $F, \mathcal{R}_0 \models Q$. We extend this algorithm to an algorithm that decides in finite time if $F, \mathcal{R}_1 \models Q$, as follows:

Data: $(F, \mathcal{R}_1 = \mathcal{R}_0 \cup \mathcal{R}_{disc}, Q)$
Result: YES iff $F, \mathcal{R}_1 \models Q$, NO otherwise.
$F' \leftarrow F$;
repeat
  forall $R_D = (H_D, C_D) \in \mathcal{R}_{disc}$ **do**
    if *Ded*$(F', \mathcal{R}_0, H_D)$ **then**
      $F' \leftarrow F' \cup \{C_D\}$ (with a renaming substitution);
      Remove R_D from $\mathcal{R}_{disc}$;
until *stability of* $\mathcal{R}_{disc}$;
return *Ded*$(F', \mathcal{R}_0, Q)$;

□

6.3. Incompatibility Results

We say that two sets of rules $\mathcal{R}_1$ and $\mathcal{R}_2$ are *equivalent* w.r.t. a vocabulary $\mathcal{V}$ if, for any fact F built on $\mathcal{V}$, the sets of facts on $\mathcal{V}$ entailed respectively by knowledge bases $(F, \mathcal{R}_1)$ and $(F, \mathcal{R}_2)$ are equals. F.i. the transformation mentioned in Section 3.2 (Example 3) transforms a set of rules into an equivalent set of rules with atomic conclusions. We consider here two other simple transformations from a rule into an equivalent pair of rules:

- τ_1 rewrites a rule $R = (H, C)$ into two rules:
  $R_h = H \rightarrow R(x_1 \ldots x_p)$ and
  $R_c = R(x_1 \ldots x_p) \rightarrow C$, where $\{x_1, \ldots, x_p\} = var(H)$ and R is a new predicate (i.e., not belonging to the vocabulary) assigned to the rule. Note that R_h is both range-restricted and domain-restricted, and R_c is atomic-hypothesis.

- τ_2 is similar to τ_1, except that the atom $R(...)$ contains all variables in the rule: $R_h = H \rightarrow R(y_1, \ldots, y_k)$ and
  $R(y_1, \ldots, y_k) \rightarrow C$, where $\{y_1, \ldots, y_k\} = \mathit{var}(R)$. Note that, among other properties, R_h is domain-restricted, while R_c is range-restricted.

Property 17. *Any set of rules can be split into an equivalent set of rules by τ_1 or τ_2.*

Proof: For τ_1 (and similarly for τ_2), we prove that, given a set of rules $\mathcal{R}$ and a fact F, both on a vocabulary $\mathcal{V}$, there is an $\mathcal{R}$-derivation F' of F iff there is a $\tau_1(\mathcal{R})$-derivation F'' of F such that the restriction of F'' to $\mathcal{V}$ is isomorphic to F'. For each part of the equivalence, the proof can be done by induction on the length of a derivation sequence. In the $\Rightarrow$ direction, it suffices to decompose each step of the $\mathcal{R}$-derivation sequence according to τ_1. In the $\Leftarrow$ direction, we show that any $\tau_1(\mathcal{R})$-derivation sequence can be reordered so that the rule applications corresponding to the application of a rule in $\mathcal{R}$ are consecutive. The reason is that the atom $R(\ldots)$ added by a rule application according to a given homomorphism keeps (at least) all information needed to apply R according to this homomorphism and cannot be used to apply another rule. □

Theorem 10. *Any instance of* ENTAILMENT *can be reduced to an instance of* ENTAILMENT *with a set of rules restricted to two rules, such that each rule belongs to a decidable class.*

Proof: From Theorem 8, any instance of ENTAILMENT can be encoded by an instance with a single rule, say R. By splitting R with τ_1 or τ_2, we obtain the wanted pair of rules. □

If we furthermore consider the concrete classes of the rules obtained by both transformations, we obtain the following result:

Theorem 11. ENTAILMENT *remains undecidable if $\mathcal{R}$ is composed of*

- *a range-restricted rule and an atomic-hypothesis rule*
- *a range-restricted rule and a domain-restricted rule*
- *an atomic-hypothesis rule and a domain-restricted rule.*

Since *ah*-rules are also *g*-rules, this implies that *g*-rules are incompatible with *rr*-rules and *dr*-rules. The case of *fr1* is more tricky. We did not find any transformation from general rules into *fr1* rules (and other rules belonging to compatible decidable classes). To prove the incompatibility of *fr1* and *rr* (Theorem 12), we use a reduction from the halting problem of a Turing Machine. This reduction transforms an instance of the halting problem into an instance of ENTAILMENT in which all rules are either *fr1* or *rr*. The compatibility of *fr1* and *dr* is an open question.

Theorem 12. ENTAILMENT *remains undecidable if $\mathcal{R}$ is composed of* fr1*-rules and* rr*-rules.*

Proof: See Appendix C. □

The following table synthesizes decidability results for the union of decidable classes based on individual criteria; ND means "not preserving decidability".

rr	fes *(wa)*					
ID/ah	fg	ND				
g	fg	ND	g			
fr1	fg	ND	fg	fg		
fg	fg	ND	fg	fg	fg	
dr	dr	ND	ND	ND	Open	ND
	disc	**rr**	**id/ah**	**g**	**fr1**	**fg**

We can also conclude for concrete classes based on global criteria, i.e., *wg*, *wfg*, *wa* and *aGRD*: all of them are incompatible, which includes the incompatibility of each class with itself (indeed, the union of two sets satisfying a global property does generally not satisfy this property; a single added rule may lead to violate any of the above global properties).

Theorem 13. *The union of two sets belonging to classes* wg*,* wfg*,* wa *and* aGRD *does not preserve decidability.*

Proof: See that the transformation τ_1 decomposes a rule into two rules R_h and R_c such that $\{R_h\}$ and $\{R_c\}$ are each *wa*, *aGRD* and *wg*. Let I be any instance of ENTAILMENT. I is transformed into an instance containing a single rule by the reduction in the proof of Theorem 8. Let I' be the instance obtained by applying τ_1 to this rule. The set of rules in I' is the union of two (singleton) sets both wa, $aGRD$ and wg. Since I' is a positive instance iff I is, we have the result. □

It follows from previous results that abstract classes are incompatible:

Theorem 14. *The union of two sets belonging to classes* fes*,* bts *or* fus *does not preserve decidability.*

Proof: Follows from Theorem 11 (for all possible pairs except *fes/fes*) and 13 (for the pair *fes/fes*). □

To conclude, the rough union of two sets of rules belonging to different decidable classes almost always leads to undecidability. The next question is whether they can be combined under some constraints. This issue is studied in the following section. We introduce the graph of rule dependencies and define conditions on the structure of this graph which constrain the interactions between rules so that decidability is preserved.

7. Combining Decidable Cases with Rule Dependencies

Generally speaking, compiling a knowledge base involves preprocessing it off-line, so that the compiled form obtained can be used on-line to accelerate reasoning tasks (e.g., query answering). Concerning rules, a classical compilation technique consists of precomputing a graph encoding dependencies between rules. In this paper, we will not detail how this technique allows one to improve the efficiency of forward and backward chaining mechanisms (as done for example in [4]), but rather use it to extend decidable classes of rules.

7.1. The Graph of Rules Dependencies (GRD)

A rule R' is said to *depend* on a rule R if the application of R on a fact may trigger a *new* application of R'.

Definition 25 (Dependency). *Let $R = (H, C)$ be a rule and Q be a fact. We say that Q* depends *on R if there is a fact F, a homomorphism $\pi : H \rightarrow F$ and a homomorphism $\pi' : Q \rightarrow \alpha(F, R, \pi)$, such that π' is not a homomorphism to F. By extension, we say that a rule R' depends on R if hyp(R') —seen as a fact— depends on R.*

The following graph encodes dependencies on a set of rules:

Definition 26 (Graph of Rule Dependencies). *Let $\mathcal{R}$ be a set of rules. The* graph of rule dependencies *(GRD) of $\mathcal{R}$ (notation GRD($\mathcal{R}$)) is a directed graph $(\mathcal{R}, E)$, where $\mathcal{R}$ is the set of nodes and E is the set of arcs, such that (R, R') is an arc of E if and only if R' depends on R.*

The facts of the knowledge base (i.e., F in the ENTAILMENT problem) can be added to the GRD as rules with an empty hypothesis and are thus sources in the GRD (i.e., nodes without incoming arcs). Similarly, a query (i.e., Q in the ENTAILMENT problem) can be added to the GRD as if it was a rule with an empty conclusion and is thus a sink in the GRD (i.e., a node without outgoing arc).

It is easy to define necessary conditions for a rule to depend on another: e.g., if R' depends on R then there is an atom in *hyp*(R') that can be unified (in the classical meaning) with an atom in *conc*(R). Characterizing dependency by actually computable necessary and sufficient conditions is less obvious. Next Theorem 15 shows that piece-unifiers allow to capture the dependency notion: a rule R' depends on a rule R if and only if there is a piece-unifier of *hyp*(R') with R, that satisfies a simple syntactic condition. This syntactic condition is not needed in the following examples and will be specified later.

Example 9 (Dependency). *Let $R_0 = p(x, y) \wedge p(y, x) \rightarrow p(x, z) \wedge p(z, t) \wedge p(t, x)$. According to the weak "atomic unification" criterion, R_0 could depend on itself. The piece-unifier criterion allows to see that it is not the case: C_0 is a single piece and, since R_0 has only one cutpoint (x), there should be a homomorphism from H_0 to C_0 (which would map x to x). Note that if z was replaced by y in C_0, the rule obtained would have 2 cutpoints and 2 pieces, and would depend on itself.*

Example 10 (GRD). *Cf. Figure 7. Let $\mathcal{R} = \{R_0, R_1, R_2, R_3\}$, where:*
$R_0 = p(x, y) \wedge p(y, x) \rightarrow p(x, z) \wedge p(z, t) \wedge p(t, x)$ *(see Example 9)*
$R_1 = q(x) \wedge p(x, y) \rightarrow q(y)$
$R_2 = p(x, y) \rightarrow r(x, y, z) \wedge p(z, w)$
$R_3 = s(x) \wedge t(x, y) \rightarrow p(x, y)$
GRD($\mathcal{R}$) has two loops (R_1, R_1) and (R_2, R_2) plus the arcs (R_0, R_1), (R_0, R_2), (R_3, R_0), (R_3, R_1) and (R_3, R_2).

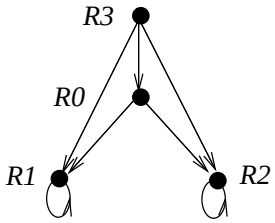


Figure 7: GRD

Let us point out that decomposing rule conclusions into single atoms (see Section 3.2) would weaken the GRD notion. Indeed, "fake" dependencies could be introduced. For instance, in Example 3 and inserting Q (as if it was a rule with an empty conclusion): there are arcs from R_1^A to R_2^A, R_3^A and R_4^A as well as arcs from R_2^A, R_3^A and R_4^A to Q (which is the bad point), whereas there would be no arc from R to Q. Thus, even if the GRD obtained encodes exactly the dependencies within $\{R_1^A, \ldots, R_4^A, Q\}$, it is not optimal w.r.t. $\{R, Q\}$.

We now precise the relation between dependency and piece-unifier.

Property 18. *If a fact Q depends on a rule R, then there is a piece-unifier of Q with R.*

Proof: See Appendix D. □

For the converse direction, one has to be more careful. Let us first consider the rule $R = p(x) \rightarrow p(x)$ and the fact $Q = p(x)$. There is a unifier of Q with R, but no application of R to a fact F can create any new atom, thus every homomorphism from Q to an $\alpha(F, R, \pi)$ is already a homomorphism from Q to F. One could remove these "obviously redundant rules", but that would not be enough. Indeed, let us now consider the rule $R = p(x) \wedge r(A, B) \rightarrow r(x, B)$ and the fact $Q = p(A) \wedge r(A, B)$. R may produce new information (for instance $r(C, B)$ when applied to the fact $p(C) \wedge r(A, B)$). There exists a unifier of Q with R, but no application of R to any fact F can create a new homomorphism from Q to $\alpha(F, R, \pi)$. This time, the redundancy is not in the rule itself; it is in the interaction between the rule and the unifier. This is that kind of redundancy we capture with the notion of atom-erasing unifier.

Definition 27 (Atom erasing unifier). *Let R be a rule and Q be a fact. We say that a unifier $\mu = (T_Q, Q', \sigma_R, \pi_Q)$ of Q with R is* atom-erasing *if there is an atom a in Q' such that $\pi_Q(a)$ is not an atom of $\beta(Q, R, \mu)$.*

Theorem 15. *A fact Q depends on a rule $R = (H, C)$ if and only if there exists an atom-erasing unifier of Q with R.*

Proof: See Appendix D. □

7.2. Decidable Cases based on the GRD

In this section, we bring out decidable cases directly based on the structure of the GRD. Let us consider the basic forward chaining mechanism, that proceeds in a breadth-first way, i.e., at each step it computes all new rule applications w.r.t. the current fact, then applies them to produce a new fact. If a subset of rules $S \subseteq \mathcal{R}$ has been applied at step i, then the only rules that have to be checked for applicability at step $i+1$ are in the set $\{R' \in \mathcal{R} | \exists R \in S, (R, R') \in E\}$. Similar arguments apply for backward chaining. The next theorem follows:

Theorem 16. *Let $\mathcal{R}$ be a set of rules. If GRD($\mathcal{R}$) has no circuit, then $\mathcal{R}$ is both a* fes *and a* fus*.*

Note that a GRD restricted to a single node with a loop, i.e., a self-unifiable rule, is sufficient to yield the undecidability of the ENTAILMENT problem (this a corollary of Theorem 8). One can however accept some kinds of circuits, as stated in the next theorem. We consider here the strongly connected components of the GRD. Let us recall that two nodes x and y in a directed graph are in the same strongly connected component of this graph if there are directed paths from x to y and from y to x. Any isolated node forms its own strongly connected component. A strongly connected component in the GRD forms a maximal set of rules that mutually depend on each other.

Theorem 17. *Let $\mathcal{R}$ be a set of rules. If all strongly connected components of GRD($\mathcal{R}$) are* fes *(resp.* fus*), then $\mathcal{R}$ is a* fes *(resp.* fus*).*

Proof: Let $C_1 \dots C_p$ be the strongly connected components of GRD($\mathcal{R}$). Assume there are all *fes*. Consider the reduced graph, say G, corresponding to these components: G has one node c_i for each C_i and an arc $c_i c_j$ iff there is an arc from a rule in C_i to a rule in C_j. By definition, G has no circuit. Associate with each c_i the maximal length of a path from a source in G to c_i. Let us call it the level of c_i and of the corresponding rule subset. We adapt the forward chaining mechanism as follows: at step 1, it processes all subsets of rules with level 0 (the sources). At step k, it processes all subsets of rules with level $k-1$ (i.e., it computes a full derivation of the current fact with these rules). This process is finite because each subset of rules is a *fes* and the number of steps is the maximal length of a path in G plus 1. It computes a full derivation because a rule of level i depends only on rules of levels less or equal to i. We conclude that $\mathcal{R}$ is a *fes*. Similar reasoning applies to the *fus* case: the rules are processed by decreasing level instead of increasing level, and processing a subset of rules consists in updating the set of the most general rewritings. □

Despite the combination of *fes* does generally not preserve decidability, the GRD allows to exhibit conditions of safe combination. The same holds for *fus*. Now, what about combining *fes*/*bts* and *fus*? This is the topic of the next section.

7.3. Combining Abstract Classes

In this section, we systematically explore decidable combinations of rule sets belonging to different abstract classes. Though the rough union of such sets generally leads to undecidability, constraints on the interactions between rules (and more specifically the following "directed cuts" in the GRD) provide us with halting algorithms for more general classes of rules.

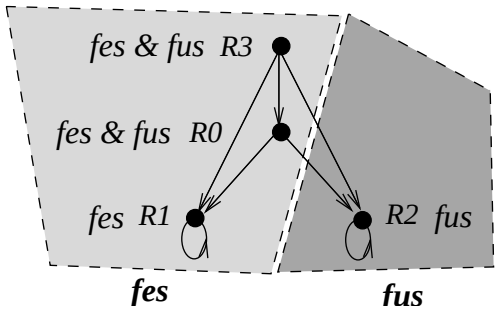


Figure 8: Set of rules in the class *fes*▷*fus*

Definition 28 (directed cut of a ruleset). *A (directed)* cut *of a set of rules $\mathcal{R}$ is a partition $\{\mathcal{R}_1, \mathcal{R}_2\}$ of $\mathcal{R}$ such that no rule in $\mathcal{R}_1$ depends on a rule in $\mathcal{R}_2$. It is denoted $\mathcal{R}_1 \triangleright \mathcal{R}_2$ ("$\mathcal{R}_1$ precedes $\mathcal{R}_2$").*

Such partitions are interesting because they allow to reason successively and independently with the two sets of rules, as shown by the following property.

Property 19. *Let $\mathcal{R}$ be a set of rules admitting a cut $\mathcal{R}_1 \triangleright \mathcal{R}_2$. Then, for any facts F and Q, it holds that $F, \mathcal{R} \models Q$ iff there is a fact P such that $F, \mathcal{R}_1 \models P$ and $P, \mathcal{R}_2 \models Q$.*

Proof: ($\Leftarrow$) Immediate. ($\Rightarrow$) Suppose that $\mathcal{R}$ admits a cut $\mathcal{R}_1 \triangleright \mathcal{R}_2$ and $F, \mathcal{R} \models Q$. Then there is a derivation sequence $\mathcal{S} = (F = F_0, F_1, \ldots, F_k)$ such that $F_k \models Q$ and each F_i is generated from F_{i-1} either by an application of a rule in $\mathcal{R}_1$ or by an application of a rule in $\mathcal{R}_2$. Since no rule in $\mathcal{R}_1$ depends on a rule in $\mathcal{R}_2$, we can reorder the rule applications such that all applications of rules in $\mathcal{R}_1$ precede all applications of rules in $\mathcal{R}_2$. More precisely, let $R = (H, C)$ and $R' = (H', C')$ be two rules s.t. R' does not depend on R; then for any fact F, for any homomorphism π from H to F and for any homomorphism π' from H' to $\alpha(F, R, \pi)$, π' is a homomorphism from H' to F, thus $\alpha(\alpha(F, R, \pi), R', \pi') = \alpha(\alpha(F, R', \pi'), R, \pi)$ [up to isomorphism]; using this property, from $\mathcal{S}$ we can build a derivation sequence $(F = F'_0, F'_1, \ldots, F'_q = P, F'_{q+1}, F'_k = F_k)$ where P is an $\mathcal{R}_1$-derivation of F and F_k is an $\mathcal{R}_2$-derivation of P. □

We will now use this property to combine rules belonging to decidable classes. For that, we define the following notations: given $\mathcal{C}_1$ and $\mathcal{C}_2$ two classes of sets of rules, a cut $(\mathcal{R}_1 \triangleright \mathcal{R}_2)$ is said to be a $\mathcal{C}_1 \triangleright \mathcal{C}_2$-cut if $\mathcal{R}_1$ belongs to the class $\mathcal{C}_1$ and $\mathcal{R}_2$ belongs to the class $\mathcal{C}_2$. The class $\mathcal{C}_1 \triangleright \mathcal{C}_2$ is the class of sets of rules that admit at least one $\mathcal{C}_1 \triangleright \mathcal{C}_2$-cut.

Example 10 (continued) Cf. Figure 8. Each rule forms its own strongly connected component. $\{R_1\}$ is a *fes* since R_1 is *rr*, but it is not a *fus*. $\{R_2\}$ is a *fus* since R_2 is *ah* (and *dr*), but it is not a *fes*. $\{R_3\}$ is a *fes* and a *fus* because R_3 is *rr* and *dr*. R_0 does not belong to one of our concrete classes, but it is a *fes* and a *fus* since its *GRD* has no arc. From Theorem 17, we conclude that $\{R_0, R_1, R_3\}$ is a *fes* and $\{R_0, R_2, R_3\}$ is a *fus*. To show that $\mathcal{R}$ is a *fes* ▷ *fus*, we have to find an appropriate cut. In such a cut,

R_3 and R_0 are necessarily in the *fes* part because of their arcs to R_1. Thus, the only *fes*$\triangleright$*fus*-cut is $\{\{R_0, R_1, R_3\}, \{R_2\}\}$.

With previous notations, Theorem 17 can be expressed as: the class *fes* $\triangleright$ *fes* is included in the class *fes*, and the class *fus* $\triangleright$ *fus* is included in the class *fus*. Theorem 12 shows that the rough union of a *fr1* set and a *rr* set can lead to undecidability. With a slight transformation of its proof (see the endnote in Appendix C), it can be shown that a Turing machine can be encoded by a set of rules of form *fr1* $\triangleright$ *rr*. Thus, the class *bts* $\triangleright$ *fes* (*a fortiori bts* $\triangleright$ *bts*) is undecidable. The following theorem provides a way to combine *fes* and *bts*.

Theorem 18 (*fes*$\triangleright$*bts*). *The class fes*$\triangleright$*bts is a subclass of* bts*.*

Proof: If $\mathcal{R}$ is *fes*$\triangleright$*bts*, then it admits a $(\mathcal{R}_1 \triangleright \mathcal{R}_2)$-cut where $\mathcal{R}_1$ is a *fes* and $\mathcal{R}_2$ is a *bts*. From property 19 and the definition of *fes*, it follows that $F, \mathcal{R} \models Q$ iff $F_k, \mathcal{R}_2 \models Q$, where F_k is a full fact equivalent to $\alpha_\infty(F, \mathcal{R}_1)$. Then, since $\mathcal{R}_2$ is a *bts*, $\mathcal{R}$ is also a *bts*. □

The variant proof of Theorem 12 (see the endnote in Appendix C) shows that an *ah*$\triangleright$*rr* rule set can lead to undecidability. Thus, the class *fus*$\triangleright$*fes* (*a fortiori fus*$\triangleright$*bts*) is undecidable. The following theorem provides a way to combine them.

Theorem 19 (*bts*$\triangleright$*fus*). ENTAILMENT *is decidable when restricted to the bts*$\triangleright$*fus class of rules.*

Proof: If $\mathcal{R}$ is *bts*$\triangleright$*fus*, then it admits an $(\mathcal{R}_1 \triangleright \mathcal{R}_2)$-cut where $\mathcal{R}_1$ is *bts* and $\mathcal{R}_2$ is *fus*. From Property 19, it follows that $F, \mathcal{R} \models Q$ iff there is an $\mathcal{R}_2$-rewriting Q' of Q s.t. $F, \mathcal{R}_1 \models Q'$. Since $\mathcal{R}_2$ is *fus*, the set $\mathcal{Q}$ of all (most general) $\mathcal{R}_2$-rewritings of Q is finite, and thus ENTAILMENT can be solved by a finite number of calls to an algorithm solving $F, \mathcal{R}_1 \models Q_i$ (where $Q_i \in \mathcal{Q}$). Each of these calls can be performed in finite time since $\mathcal{R}_1$ is *bts*. □

Note that in the specific case of a *fes*$\triangleright$*fus* set provided with an appropriate cut, say $(\mathcal{R}_1, \mathcal{R}_2)$, we have an effective sound and complete halting mechanism. Indeed, we can on the one hand use forward chaining on $\mathcal{R}_1$ to compute a full derivation of the facts, say F', on the other hand use backward chaining on $\mathcal{R}_2$ to compute the finite set $\mathcal{Q}$ of most general rewritings of Q, then check if there is an element of $\mathcal{Q}$ that maps to F'.

Figure 9 summarizes decidable combinations of $\forall\exists$-rule sets. An arrow from a class A to a class B means "there can be dependency arcs from rules in A to rules in B", and no arrow from A to B means "no dependency arc can exist from a rule in A to a rule in B". We can recursively build a *fes* from a *fes*$\triangleright$*fes*, or a *fus* from a *fus*$\triangleright$*fus* (Theorem 17): in the picture, (3) is built from (1) and (2), and (12) is built from (11) and (10). This theorem does not hold for *bts*, but the union of two *bts* without any dependency arc from one class to the other is still a *bts*: (7) is a *bts* obtained from (5) and (6). The rough union with disconnected rules preserves *fes*, *bts* and *fus* (see the proof of Theorem 9): the *fes* (4), the *bts* (8) and the *fus* (13) are built respectively from (3), (7) and (12) by a rough union with disconnected rules. We can obtain a *bts* from a *fes*$\triangleright$*bts* (Theorem 18): the *bts* (9) is obtained from (4) and (8). Finally, both *fes*$\triangleright$*fus* and *bts* $\triangleright$ *fus* are decidable (Theorem 19): the *fes* $\triangleright$ *fus* (14) is obtained from (4) and (13); and the *bts*$\triangleright$*fus* (15) is obtained from (9) and (13).

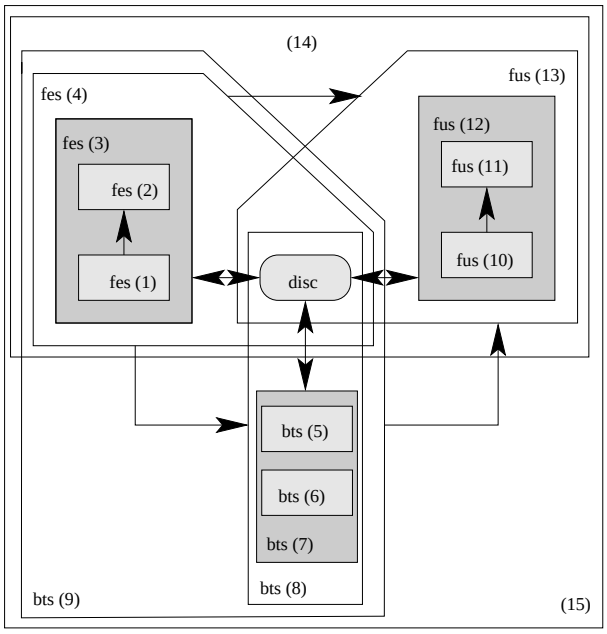


Figure 9: Dependency-based decidable combinations of rules

8. Rules with Equality and Negative Constraints

In this section, we consider rules with equality as well as negative constraints and integrate them into our framework.

8.1. Equality Rules

When it occurs in rule hypotheses, equality either makes these rules unapplicable (case of an equality between distinct constants, which contradicts the unique name assumption) or can be suppressed with a simple transformation of the knowledge base. In a rule conclusion, an equality of the form $x = t$, where x is an existentially quantified variable, can be equivalently replaced by substituting x by t. Hence, let us focus on equality occurring in a rule conclusion and involving only variables shared with the hypothesis or constants (i.e., cutpoints). Let $R = (H, C)$ be such a rule, with C being composed of a non-empty set of regular atoms, say C', and a set of equality atoms $e_1 = e'_1, \ldots, e_n = e'_n$, where all e_i and e'_i are distinct cutpoints in R. R can be equivalently rewritten as a rule (H, C') and one rule for each equality, i.e., n rules of form $(H, e_i = e'_i)$, $1 \leq i \leq n$ (note that each equality atom forms its own piece in R). Moreover, due to the unique name assumption, an equality between two distinct constants leads to the inconsistency of the fact to which the rule should be applied: the rule then acts as a negative constraint, that we will consider in Section 8.2. For these reasons, we focus our attention on the following rules, called *equality rules*:

Definition 29 (Equality rule). *An equality rule $R = (H, x = t)$ is a formula of form $\forall x_1 \dots \forall x_p (H \rightarrow x = t)$, where x and t are distinct terms, $x \in var(H)$ and $t \in var(H)$ or is a constant. If t is a variable (resp. a constant): R is applicable to a fact F if there is a homomorphism, say π, from H to F with $\pi(x) \neq \pi(t)$ (resp. $\pi(x) \neq t$). If $\pi(x)$ and $\pi(t)$ (resp. t) are distinct constants, then the application fails; the application of R to F according to π is as follows: if $\pi(x)$ is a variable, each occurrence of $\pi(x)$ in F is replaced by $\pi(t)$ (resp. t); otherwise ($\pi(x)$ is a constant and t and $\pi(t)$ are variables), each occurrence of $\pi(t)$ is replaced in F by $\pi(x)$.*

Note that when an application fails, the knowledge base is logically inconsistent (due to the unique name assumption, see Definition 1), and reciprocally. Indeed, the application of $(H, x = y)$ fails if and only if there are two distinct constants a and b such that $\sigma(H)$ is entailed by the KB, where $\sigma = \{(x, a), (y, b)\}$. A similar construction can be done for equality rules with a constant in the conclusion. It is immediate to check that ENTAILMENT and consistency checking are reducible to each other for KB containing both $\forall\exists$-rules and equality rules.

Equality rules generalize functional dependencies, which are widely used in data modeling and ontologies. A functional dependency expresses that, for a given relation (i.e., predicate), for a given sublist l of its arguments, each value for l uniquely defines the value of a given argument outside l. For instance, a functional dependency expressing that, for a ternary relation r, the value of the two first arguments determines the value of the third, is translated by the equality rule $\forall x \forall y \forall z_1 \forall z_2 (r(x, y, z_1) \wedge r(x, y, z_2) \rightarrow z_1 = z_2)$.

The bad news is that almost all rules depend on a rule with equality. It is easy to check that a rule whose hypothesis does not contain two occurrences of the same term does not depend on an equality rule of form $x = y$, where y is a variable, and it does not depend on an equality rule of form $x = c$, where c is a constant, if moreover its hypothesis does not contain c. Otherwise, the rule potentially depends on any equality rule.

How do equality rules fit in our decidability map? Equality rules are *fes* rules. Note however that adding an equality rule to a *fes* does not guarantee that this set remains a *fes*, as shown in the next example. It even leads to undecidability, as shown in Theorem 20.

Example 11 (*fes* and equality rules). *Let $\mathcal{R} = \{r(x, x) \rightarrow s(x, y) \wedge r(y, z)\}$. Let $R_e = s(x, y) \wedge r(y, z) \rightarrow y = z$. $\mathcal{R}$ is a* fes *but $\mathcal{R} \cup \{R_e\}$ is not a* fes.

Equality rules satisfy the range-restricted property, i.e., $var(C) \subseteq var(H)$. They can thus be safely added to *rr*-rules. Furthermore, from the algorithm in the proof of Theorem 9 ("universal compatibility" of disconnected rules) it follows that the property of being a *fes* is kept when disconnected rules are added. Hence, the following property:

Property 20. *The union of sets of* rr*-rules,* disc*-rules and equality rules is a* fes.

The previous property cannot be generalized to any set of $\forall\exists$-rules belonging to the *fes* class, as shown by the next theorem.

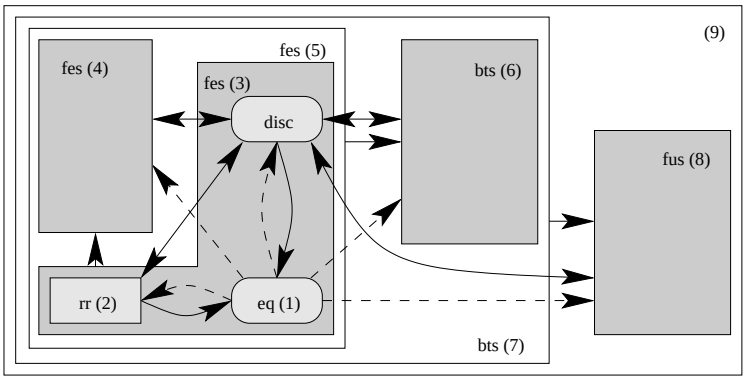


Figure 10: Dependency-based decidable combinations of rules and equality rules

Theorem 20. *The addition of one equality rule to a* fes *(*a fortiori *a* bts*) or a* fus *does not preserve the decidability.*

Proof: We transform any $\forall\exists$-rule into an equivalent set of three rules, such that the first two rules form a *fes* and a *fus* and the third one is an equality rule. We conclude with Theorem 8, which states that ENTAILMENT remains undecidable with a single rule. The transformation is as follows. Let $R = (H, C)$ be any $\forall\exists$-rule, and let R_1, R_2 and R_3 be the obtained rules. Similar to the transformations in Section 6.3, we assign to R a new predicate, say R, with arity $|var(H)| + 2$.
$R_1 = H \rightarrow R(x_1, \ldots, x_p, x, y)$, where $\{x_1, \ldots, x_p\} = var(H)$, and x, y are variables not occurring in H.
$R_2 = R(x_1, \ldots, x_p, x, x) \rightarrow C$, where x is a variable not occurring elsewhere in R_2.
$R_3 = R(x_1, \ldots, x_p, x, y) \rightarrow x = y$. Note that R_2 cannot be applied directly after R_1; an application of R_3 is needed first. The proof of the equivalence of $\{R\}$ and $\{R_1, R_2, R_3\}$ is similar to the proof of Property 17. To see that $\{R_1, R_2\}$ is a *fes* and a *fus*, we build the associated GRD: the only possible arc in this GRD is from R_2 to R_1 (R_1 depends on R_2 if R depends on itself); the GRD has no circuit, thus $\{R_1, R_2\}$ is a *fes* and a *fus* (Theorem 16). □

In Figure 10, we complete the picture of decidable combinations of rules provided in Figure 9 by introducing equality rules. Note that *fes* (4), *bts* (6) and *fus* (8) can be built as described in Figure 9. In this drawing, the assumption is that all rule components depend on the *eq* set of rules (1) (these dependencies are represented by dashed arrows). As stated by Property 20, the rough union of disconnected rules (*disc*), equality rules (1) and range-restricted rules (2) is a *fes* (3 in the drawing). This *fes* can then be combined with another *fes* to form a *fes*: $(5) = (3) \triangleright (4)$. Finally, the obtained set can be combined as in Figure 9 with *bts* and *fus* to yield larger decidable classes.

8.2. Negative Constraints

A negative constraint expresses that a specific fact, say C, should not be entailed by the knowledge base. It is often defined as a rule of form $(C, \bot)$, where $\bot$ denotes

the absurd symbol (i.e., a propositional atom whose value is false). Equivalently, it can be defined as $\neg C$ as in [8]. It is satisfied by a fact F if $F \not\models C$. It is satisfied by a knowledge base $\mathcal{K} = (F, \mathcal{R})$ if $\mathcal{K} \not\models C$, i.e., no fact to which C maps can be $\mathcal{R}$-derived from F. A typical use of negative constraints is to express disjointness of concepts/classes or incompatibility of relations.

Negative constraints can be integrated in the previous framework without any difficulty. Let us consider knowledge bases of form $(F, \mathcal{R}, \mathcal{C})$, where F is a fact, $\mathcal{R}$ is a set of $\forall\exists$-rules and $\mathcal{C}$ is a set of negative constraints. Such a knowledge base is consistent if $(F, \mathcal{R})$ satisfies each constraint in $\mathcal{C}$. A fact Q is entailed by $\mathcal{K} = (F, \mathcal{R}, \mathcal{C})$ if and only if either $\mathcal{K}$ is inconsistent (and then it entails everything) or $(F, \mathcal{R}) \models Q$. Checking the entailment of Q can thus be solved by $|\mathcal{C}| + 1$ calls to an algorithm solving the ENTAILMENT problem on KBs without constraints. Now, if a set of equality rules, say $\mathcal{E}$, is also considered, another source of inconsistency is added and one has furthermore that check that $(F, \mathcal{R} \cup \mathcal{E})$ is consistent.

9. Related Work

The framework studied in this paper is closely related to research works on tuple-generating dependencies in databases and on conceptual graph rules. This section is devoted to these connections.

Tuple- and Equality-generating dependencies. Tuple-generating dependencies (TGDs) and equality-generating dependencies (EGDs), introduced in [11], generalize the main classes of dependencies that were previously studied in database systems. A TGD has the same form as a $\forall\exists$-rule. It is considered as a constraint to be satisfied by database instances. When a TGD is not satisfied by a database instance, say D, it is possible to repair D by extending it with new atoms. The procedure that enforces the validity of a set of TGDs is called the *chase*: it is equivalent to forward chaining. The chase was first introduced for the *TGD implication problem*: given a set of TGDs T and a TGD t, is t implied by T? This problem is the same as our RULE ENTAILMENT problem (Section 2.4). A related problem is the *query containment problem* under a set of TGDs: given a set of TGDs T and two conjunctive queries q_1 and q_2, is the the set of answers to q_1 included in the set of answers to q_2 for any database satisfying T (i.e., satisfying each TGD in T)? This problem is the same as our FACT ENTAILMENT problem (Section 2.4), with the KB (q_1, T) and the query q_2. A problem introduced more recently is *query answering on incomplete data* [17]: given a set of TGDs T, a database instance D, that may not satisfy T, a conjunctive query q and a tuple of values t, is t an answer to q in a database instance obtained from D by enforcing T? This problem is the same as our QUERY EVALUATION problem (Section 2.4).

All decidable cases exhibited for TGDs are based on the chase procedure. Until recently, all these cases were based on the finiteness of the chase (thus belong to the *fes* abstract class), with the exception of the pioneer paper of [32] that showed that query containment on inclusion dependencies (ID) is decidable even if the chase may not halt.

[12] generalized previous classes with a decidability condition based on the finite treewidth model property, which we use to define the *bts* abstract class, and introduced

the guarded and weakly-guarded classes, which are concrete *bts* classes. In this paper, we enrich the picture by generalizing these classes and defining decidable cases based on the finiteness of backward chaining (which yields the *fus* abstract class).

[26] studies "sets of TGDs with universal models", which are exactly *fes*. The "chase graph" is equivalent to our GRD, but no constructive characterization of this graph is provided in [26]. The main decidable case is the "stratified chase graph", which can be seen as an instantiation of the *fes* part of Theorem 17 with weakly acyclic TGDs: if all strongly connected components of the GRD are weakly acyclic, then the problem is decidable.

EGDs can be seen as equality rules without constant in the conclusion. It has long been shown that the interaction between EGDs and simple cases of TGDs leads to undecidability of ENTAILMENT. This has been shown for functional dependencies (which are specific EGDs) and inclusion dependencies (which are specific TGDs) [21] ; the result still holds if functional dependencies are further restricted to key dependencies [17]. In [13] an abstract condition called *separability* (generalizing the non-key-conflicting notions introduced in [17] and [12]) is defined on the set $T \cup E$, where T is a set of TGDs and E is a set of EGDs. This condition allows to process E separately from T: briefly, E can then be considered as a set of constraints to be satisfied by the initial database instance and the entailment (or query answering) considers T only. A syntactic condition sufficient to ensure separability for EGDs representing functional dependencies is also given (note however that it ensures separability only if the conclusions of TGDs are restricted to a single atom). Our results (Property 20 and Figure 10) provide some cases where the chase can process EGDs triggered by TGDs.

Query answering over ontologies. Guarded rules, and their generalization to frontier-guarded rules, allow to encode some recent DLs tailored for query answering, as we pointed it out in Section 5.2. A general framework dedicated to conjunctive query answering with ontologies is proposed in [13]. This Datalog-based framework is called Datalog$^{\pm}$. On one hand, this framework extends Datalog with TGDs, EGDs and negative constraints, on the other hand it restricts TGDs and EGDs to achieve decidability and tractability. Our decidable classes can be seen as defining new members of this family.

Conceptual Graph rules. Conceptual graphs [36] are a family of graphical languages with a FOL semantics. The kernel formalism, called basic conceptual graphs, is equivalent to the existential positive conjunctive fragment of FOL, thus corresponds to facts and conjunctive queries. Its extension to graph rules is equivalent to the $\forall\exists$-rule fragment [23]. Basic conceptual graphs are defined on a lightweight ontology essentially composed of partially ordered sets of concepts (unary predicates) and relations (n-ary predicates). This ontology could be encoded by simple range-restricted rules (with $t_1 \leq t_2$ being translated into $\forall x_1 \ldots \forall x_k(t_1(x_1, \ldots, x_k) \rightarrow t_2(x_1, \ldots, x_k))$, where $k = 1$ if t_1 and t_2 are concepts, otherwise k is the arity of the relations t_1 and t_2). However, concept and relation comparisons are integrated into the conceptual graph homomorphism and unifier notions, which allows for algorithmic optimizations based on the compilation of the partial orders. All results in this paper can be applied to conceptual graph rules, whatever the way of taking the partial orders into account is.

An interesting issue would be to study how the integration of specific rules encoding partial orders in the reasoning mechanisms could be used to extend our decidability results.

In turn, present results find their roots in early work of the authors on conceptual graphs. The forward chaining and the backward chaining mechanisms were essentially introduced in [35] for conceptual graph rules; their soundness and completeness were proven with different proofs. Note that an adaptation of this backward chaining to TGDs has been proposed in [24] as an alternative to the chase. A precursor of the GRD is introduced in [4]: this paper defines a criterion of dependency for CG rules, which, translated into $\forall\exists$-rules, is optimal for rules without constants. Instead, we rely on unification, which has the additional advantage of relating the notion of dependency to backward chaining. *Fes* are introduced in [8] and a result equivalent to the *fes* part in Theorem 17 is proven in [4]. The decidability and complexity of reasoning with conceptual graph rules and constraints, in particular negative constraints, is studied in [8].

10. Conclusion

In this paper, we have "walked the decidability line" for the ontological conjunctive query answering problem based on $\forall\exists$-rules: on the one hand we have extended the map of known decidable cases, and on the other hand we have brought out some negative results, in particular that the rough union of decidable classes of rules is generally not decidable. We have also shown that the graph of rule dependencies is a powerful tool for combining decidable paradigms while keeping decidability. More specifically, the main contributions of this paper are the following:

- the classification of known decidable classes, guided by three abstract classes (*fes*, *bts* and *fus*), as well as an extension of several concrete decidable classes, relying upon the notion of the *frontier* of a rule;
- a precise study of safe interactions between decidable classes and new decidability results based on the graph of rule dependencies, especially allowing to combine both forward and backward chaining mechanisms.

This paper focuses on decidability issues. Contrarily to *fes* and *fus* abstract classes, the *bts* abstract class is not provided with an effective decision procedure. Moreover, halting algorithms are known for some concrete *bts* classes, namely guarded and weakly-guarded classes [12], but it is not the case for the newly exhibited *bts* classes. Besides their intrinsic interest, such algorithms could then be used to generate halting algorithms for wider classes obtained with our combination results based on the graph of rule dependencies.

Regarding the complexity of ENTAILMENT, previously known concrete decidable classes have been classified. ENTAILMENT with range-restricted and/or disconnected rules is NP-complete if the arity of the predicates is bounded [8], otherwise the range-restricted case becomes EXPTIME-complete (from results on Datalog, e.g., [19]). ENTAILMENT with guarded rules is EXPTIME-complete with bounded predicate arity,

otherwise it is 2EXPTIME-complete, and the same complexities hold for the generalization to the weakly-guarded class [12]. ENTAILMENT with atomic-hypothesis rules is PSPACE-complete [14]. Two complexity measures are classically considered for query problems: the usual complexity, called combined complexity, and data complexity. With combined complexity, all components of the problem instance, here $\mathcal{K} = (F, \mathcal{R})$ and Q, are considered as input. With data complexity, only the data, here F, are considered as part of the input, with $\mathcal{R}$ and Q being considered as fixed. For instance, checking homomorphism from a query to a fact is NP-complete in combined complexity and polynomial in data complexity. The latter complexity is relevant when the data size is much larger than the size of the rules and the query. For the classes *rr+disc* and *g*, ENTAILMENT has polynomial data complexity, but is still EXPTIME-complete for *wg* [14]. The complexity study for *fr1*, *fg* and *wfg* is ongoing work. In particular, the question is how they behave w.r.t. to *g* and *wg* (note that the arguments used in the complexity proofs for the latter rule classes need to be generalized in a non-trivial way for *fr1* and *fg*). By definition, ENTAILMENT has polynomial data complexity with any *fus* class.

Further work also includes algorithmic optimizations exploiting the graph of rule dependencies. The first, obvious, point is that when computing the saturation at rank k, we only need to test rules that depend on rules applied at rank $k-1$; the same kind of improvement can be applied to backward chaining. However, unifiers can be used more efficiently, as shown in [9] for forward chaining. Basically, arcs (R_i, R_j) in the GRD can be labeled by all unifiers μ_k of R_j with R_i. Then, when R_i is applied to a fact F according to a homomorphism π, we can compose π with unifiers μ_k to obtain, in linear time, *partial homomorphisms* π_k of $hyp(R_j)$ to $F = \alpha(F, R_i, \pi)$; any homomorphism from $hyp(R_j)$ to F is an extension of one of these π_k. The GRD thus provides us with a partial compilation of the forward chaining mechanism. Transposition of this idea to backward chaining would require to compute *partial unifiers* from the composition of two unifiers, and this notion remains to studied more precisely.

Another research direction is the further extension of decidable classes. We have pointed out the interest of precisely studying interactions between rules to extend decidable cases. Two techniques for encoding these interactions can be found in the literature and have been mentioned above: one relies on the graph of rule dependencies and the other on a graph of position dependencies. Both graphs encode different kinds of interactions between rules. The generalization of both techniques is ongoing work.

Acknowledgements. We thank Georg Gottlob for the reference to [37], Michaël Thomazo for his careful reading of a previous version of this paper, as well as the anonymous reviewers for their relevant and constructive comments.

A. Soundness and Completeness of Homomorphism-based Mechanisms

This section is devoted to the semantics of facts and rules and proves the logical soundness and completeness of the homomorphism-based mechanisms defined for facts and rules. We first give some properties related to semantic interpretations of facts.

Definition 30. *[Witness of a fact in an interpretation] Let F be a (possibly infinite) fact on $\mathcal{V}$ and $I = (\Delta, .^I)$ be an interpretation of $\mathcal{V}$. A* witness *of F in I is a mapping $\pi : term(F) \rightarrow \Delta$ such that:*

- *for every constant $c \in const(F)$, $\pi(c) = c^I$;*
- *for every atom $p(t_1, \ldots, t_k) \in F$, $(\pi(t_1), \ldots, \pi(t_k)) \in p^I$.*

Property 21. *[Model of a fact] Let F be a (possibly infinite) fact on $\mathcal{V}$ and I be an interpretation of $\mathcal{V}$. Then I is a model of F if and only if there exists a witness of F in I.*

Proof: If I is a model of F, let v be a valuation of the existential variables that makes F true for I. Then a witness π is obtained as follows: for all $e \in term(F)$, $\pi(e) = e^I$ if e is a constant, otherwise $\pi(e) = v(e)$. Reciprocally, given a witness π, any interpretation I of $\mathcal{V}$ such that for all $c \in const(F)$, $c^I = \pi(c)$ and for all $p \in pred(F)$, $\{(\pi(e_1), \ldots, \pi(e_k)) | p(e_1, \ldots, e_k) \in F\} \subseteq p^I$ is a model of F. □

Definition 31 (Encoding interpretations as facts). *Let $I = (\Delta, .^I)$ be an interpretation of a vocabulary $\mathcal{V} = (\mathcal{P}, \mathcal{C})$. We encode I as a (possibly infinite) fact $\phi(I)$ as follows:*

- *Let ϕ be a bijective mapping from Δ to $\mathcal{C} \cup X$, where X is a set of variables, such that for all $\delta \in \Delta$, if δ is the interpretation of a (unique, thanks to the unique name assumption) constant $c \in \mathcal{C}$ (i.e., $c^I = \delta$), then $\phi(\delta) = c$.*
- *Let C be the conjunct obtained as follows: for every predicate $p \in \mathcal{V}$ and for every tuple $(\delta_1, \ldots, \delta_k) \in p^I$, $p(\phi(\delta_1), \ldots, \phi(\delta_k))$ is an atom in C;*
- *$\phi(I)$ is the existential closure of C.*

Given a fact F and an interpretation I, I is a model of F if and only if there is a homomorphism from F to $\phi(I)$. The next lemma shows how to build a witness of F in I from a homomorphism from F to $\phi(I)$, and conversely.

Lemma 3. *Let F be a fact, $I = (\Delta, .^I)$ be an interpretation and $\phi(I)$ be an encoding of I.*

1. *Let π be a mapping from term(F) to Δ. Then π is a witness of F in I if and only if $\phi \circ \pi$ is a homomorphism from F to $\phi(I)$.*
2. *Let π be a mapping from $term(F)$ to term$(\phi(I))$. Then π is a homomorphism from F to $\phi(I)$ if and only if $\phi^{-1} \circ \pi$ is a witness of F in I.*

Proof: Follows immediately from the definitions. □

Lemma 4 (Isomorphic model). *For every (possibly infinite) fact F, there exists (at least one) interpretation I such that F is an encoding of I (notation $I \in \phi^{-1}(F)$). Furthermore, I is a model of F. It is called an* isomorphic model *of F.*

Proof: Let us build I. The domain of I is $\Delta = \mathit{term}(F)$. For each $c \in \mathit{const}(F)$, we define $c^I = c$. For each $p \in pred(F)$, $p^I = \{(t_1, \ldots, t_k) | p(t_1, \ldots, t_k) \in F\}$. It can be immediately checked that F is an encoding of I. Since $F \in \phi(I)$, there is a witness of F in I (Lemma 3. 2), thus I is a model of F. □

We can now prove the soundness and completeness of homomorphism in the logical fragment of facts:

Theorem 1. *Let F and F' be two facts. $F' \models F$ if and only if there is a homomorphism from F to F'.*
Proof: ($\Leftarrow$) Assume there is a homomorphism, say π from F to F'. Let us prove that any model of F' is also a model of F. Consider I a model of F'. Then there is a witness π' from F' to I. Thus $\phi \circ \pi'$ is a homomorphism from F' to an encoding $\phi(I)$ of I (Lemma 3. 1). Since $\pi \circ \phi \circ \pi'$ is a homomorphism from F to $\phi(I)$, there exists a witness of F in I (Lemma 3. 2). ($\Rightarrow$) Assume $F' \models F$, i.e., every model of F' is a model of F. In particular, consider an isomorphic model I of F' (from Lemma 4 it exists). I is also a model of F. Since F' is an encoding of I, there is a homomorphism from F to F' (Lemma 3. 1). □

We now focus on the semantics of rules and prove that the saturation scheme is sound and complete in the fragment of facts and rules.

Property 22 (Models of a $\forall\exists$-rule). *Let $R = (H, C)$ be a $\forall\exists$-rule on $\mathcal{V}$, and $I = (\Delta, .^I)$ be an interpretation of $\mathcal{V}$. Then I is a model of R if and only if, for every witness π of H in I, there exists a witness π' of C in I such that, for all $t \in fr(R)$, $\pi(t) = \pi'(t)$ (equivalently: π can be extended to a witness of C).*

Proof: Follows from the definition of a $\forall\exists$-rule. □

Lemma 5. *Let I be a model of F and of all rules in a set $\mathcal{R}$. Then, for any integer k, I is a model of $\alpha_k(F, \mathcal{R})$.*

Proof: We prove inductively that, for any integer k, there is a witness of $\alpha_k(F, \mathcal{R})$ in any such model I. There is a witness π_0 of F in I. We prove by induction that if there is a witness π_i of $F^i = \alpha_i(F, \mathcal{R})$ in I, then there is a witness π_{i+1} of F^{i+1} in I. The induction hypothesis is satisfied at rank 0 since $F = F^0$. We have $F^{i+1} = F^i \cup_{(R=(H,C),\pi) \in \Pi(\mathcal{R}, F^i)} C'$, where $C' = \pi^{safe}(C)$. See that for every C', there exists a witness π' of C' in I such that, for all $x \in \mathit{var}(C') \cap var(F^i)$, $\pi'(x) = \pi_i(x)$ (from Property 22). Since none of the C' share any new variable (thanks to the safe substitution), the mapping obtained from π_i and all π' is a witness from F^{i+1} to I. □

Lemma 6. *Let F be a fact and $\mathcal{R}$ be a set of $\forall\exists$-rules. Then an isomorphic model I of $\alpha_\infty(F, \mathcal{R})$ is a model of F and of all rules in $\mathcal{R}$.*

Proof: Since $F \subseteq F^* = \alpha_\infty(F, \mathcal{R})$, there is a homomorphism from F to F^* and thus $F^* \models F$ (Theorem 1). Consider now any rule $R = (H, C)$ in $\mathcal{R}$ and any witness π_H of H in I (if there is none, I is a model of R). From Lemma 3.1, there is a homomorphism $\pi = \sigma \circ \pi_H$, with $\sigma(I) = F^*$ from H to $F^* = \alpha_\infty(F, \mathcal{R})$. >From Property 1, there exists k such that π is a homomorphism from H to $F^k = \alpha_k(F, \mathcal{R})$. By definition of a rule application, there is a homomorphism π' from C to $F^{k+1} = \alpha_{k+1}(F, \mathcal{R})$, thus to F^*. From Lemma 2, this homomorphism yields a witness of C in I. □

Theorem 2. *Let F and F' be two facts, and $\mathcal{R}$ be a set of $\forall\exists$-rules. Then $F, \mathcal{R} \models F'$ if and only if there is a homomorphism from F' to $\alpha_\infty(F, \mathcal{R})$.*
Proof: Let us consider F, F' and $\mathcal{R}$. Let $F^* = \alpha_\infty(F, \mathcal{R})$.

($\Leftarrow$) Assume there is a homomorphism, say π, from F' to F^*. Then there is an integer k such that π is a homomorphism from F' to $F^k = \alpha_k(F, \mathcal{R}) \subseteq F^*$. Let us prove that any model I of F and of all rules in $\mathcal{R}$ is also a model of F'. I is a model of F^k (Lemma 5) and there is a homomorphism π' from F^k to an encoding $\phi(I)$ of I (Lemma 3.1). Since $\pi' \circ \pi$ is a homomorphism from F' to $\phi(I)$, there is a witness of F' in I (Lemma 3.2), thus I is a model of F' (Property 21).

($\Rightarrow$) Assume $F, \mathcal{R} \models F'$. In particular, consider an isomorphic model I of F^*. From Lemma 6, it is a model of F and of all rules in $\mathcal{R}$. It is thus a model of F'. Since F^* is an encoding of I, there is a homomorphism from F' to F^* (Lemma 3.1).

□

B. Proof of Theorem 3

Notation. Let $\sigma : X \to T$ and $\sigma' : X' \to T'$ be two substitutions such that, $\forall t \in X \cap X', \sigma(t) = \sigma'(t)$. Then we note $\sigma + \sigma' : X \cup X' \to T \cup T'$ the substitution defined by: if $t \in X, (\sigma + \sigma')(t) = \sigma(t)$, otherwise $(\sigma + \sigma')(t) = \sigma'(t)$.

The following property, which will be used in the next lemmas, illustrates the "independence" between pieces.

Property 23. *Let F and Q be two facts, $X \subseteq var(Q)$, $Q_1, \ldots, Q_k$ be a partition of the atoms of Q such that each Q_i is the union of one or more pieces of Q according to X, and $\pi_1, \ldots, \pi_k$ are homomorphisms from the Q_i to F such that, $\forall t \in X$, $\forall 1 \leq i \leq j \leq k$, $\pi_i(t) = \pi_j(t)$; then the substitution $\pi_1 + \ldots + \pi_k$ is a homomorphism from Q to F.*

Proof: We prove the property for $k = 2$, then apply this result $k - 1$ times to prove the general result. If π_1 and π_2 are homomorphisms respectively from Q_1 and Q_2 to F, then $\pi_1 + \pi_2$ is a substitution (since the only shared variables are in X, $\pi_1 + \pi_2$ is a mapping), and it is immediate to check that it is a homomorphism. □

In the following, when several safe substitutions according to a substitution σ are involved, we denote them by $\sigma^{safe.1}, \ldots, \sigma^{safe.k}$.

Lemma 7. *Let $\mu = (T_Q, Q', \sigma_R, \pi_Q)$ be a unifier of a fact Q with a rule $R = (H, C)$. Consider $F = \beta(Q, R, \mu) = {\sigma_R}^{safe.1}(H) \cup \pi_Q(Q \setminus Q')$. We note $\alpha\beta(Q, R, \mu) = \alpha(F, R, {\sigma_R}^{safe.1})$. Then there is a homomorphism from Q to $\alpha\beta(Q, R, \mu)$.*

Proof: We build a particular homomorphism π from Q to $F' = \alpha\beta(Q, R, \mu)$, that we will later call the *natural homomorphism from Q to $\alpha\beta(Q, R, \mu)$*. We have $F' = {\sigma_R}^{safe.1}(H) \cup \pi_Q(Q \setminus Q') \cup {\sigma_R}^{safe.2}(C)$. Let us consider the two following substitutions:

- π_1 : *var*$(Q \setminus Q')$ $\rightarrow$ *terms*(F') such that $\pi_1(x) = \pi_Q(x)$ if $x \in T_Q$ and $\pi_1(x) = x$ otherwise;
- $\pi_2 =$: *var*(Q') $\rightarrow$ *terms*(F') such that $\pi_2(x) = {\sigma_R}^{safe.2}(\pi_Q(x))$.

Both π_1 and π_2 are homomorphisms and, since they verify the conditions of Property 23 (with $T_Q = X$), $\pi = \pi_1 + \pi_2$ is a homomorphism from Q to F'. □

Lemma 8. *Let F be a fact, $R = (H, C)$ be a rule and π be a homomorphism from H to F such that $F' = \alpha(F, R, \pi) = F \cup \pi^{safe.1}(C)$ contains an atom that is not in F. Then there is a unifier $\mu = (T_{F'}, \pi^{safe.1}(C), \pi, id)$ of F' with R defined by: $T_{F'} = \pi^{safe.1}(fr(R))$, and id is the identity. We note $\beta\alpha(F, R, \pi) = \beta(F', R, \mu)$. Then there is a homomorphism from $\beta\alpha(F, R, \pi)$ to F.*

Proof: First check that μ is a unifier ($\pi^{safe.1}(C)$ is not empty). Then we build a particular homomorphism π' from $F'' = \beta\alpha(F, R, \pi)$ to F, that we will later call the *natural homomorphism from $\beta\alpha(F, R, \pi)$ to F*. We have $F'' = \pi^{safe.2}(H) \cup \pi_Q((F \cup \pi^{safe.1}(C)) \setminus \pi^{safe.1}(C)) = \pi^{safe.2}(H) \cup G$, where $G \subseteq F$. Let us now consider the following substitutions:

- π_1 : *var*$(\pi^{safe.2}(H))$ $\rightarrow$ *terms*(F) such that $\forall t \in H$, $\pi_1(\pi^{safe.2}(H)) = \pi(H)$;
- π_2 : *var*(G) $\rightarrow$ *terms*(F) such that $\forall t \in$ *var*(G), $\pi_2(t) = t$.

Both π_1 and π_2 are homomorphisms and, since they verify the conditions of Property 23, then $\pi' = \pi_1 + \pi_2$ is a homomorphism from F'' to F. □

The following property (as well as its corollary) will also be used in the main theorem proof.

Property 24. *If $Q' = \alpha(Q, R, \pi_\alpha)$ and there is a homomorphism π from Q to F, then there is a homomorphism π' from Q' to $F' = \alpha(F, R, \pi \circ \pi_\alpha)$.*

Proof: With $R = (H, C)$, we have $Q' = Q \cup {\pi_\alpha}^{safe.1}(C)$ and $F' = F \cup (\pi \circ \pi_\alpha)^{safe.2}(C)$. Let us build the homomorphism π' from Q' to F'. We first consider the homomorphism π_C from ${\pi_\alpha}^{safe.1}(C)$ to $(\pi \circ \pi_\alpha)^{safe.2}(C)$ defined by:

- if $t \in \pi_\alpha($*cutp*$(R))$, then $\pi_C(t) = \pi(t)$.
- otherwise $\pi_C(t) = (\pi \circ \pi_\alpha)^{safe.2}(({\pi_\alpha}^{safe.1})^{-1}(t))$.

Since π and π_C satisfy the condition of property 23 (with $X = \pi_\alpha($*cutp*$(R))$), then $\pi' = \pi + \pi_C$ is a homomorphism from Q' to F'. □

With an immediate recursion on the previous property, we obtain:

Corollary 1. *If Q' is an $\mathcal{R}$-derivation of Q and Q maps to F, then there exists an $\mathcal{R}$-derivation F' of F such that Q' maps to F'.*

Theorem 3. *Let $K = (F, \mathcal{R})$ be a KB and Q be a fact. Then $F, \mathcal{R} \models Q$ if and only if there is an $\mathcal{R}$-rewriting of Q that maps to F.*

Proof:

($\Rightarrow$) The first direction is proven by induction on the length of the derivation sequence (*i.e.* the number of rule applications used in the derivation), and relies upon Lemma 8. The induction property is trivially true at rank 0. Let us assume it is true at rank n. Suppose that F_{n+1} is obtained from F by an $\mathcal{R}$-derivation sequence $F = F_0, F_1 = \alpha(F_0, R, \pi_\alpha), \ldots, F_{n+1}$ of length $n+1$, and Q maps to F_{n+1}, where F_{n+1} is obtained from F_1 by a derivation sequence of length n. By induction hypothesis, there exists an $\mathcal{R}$-rewriting Q_1 of Q such that there exists a homomorphism π_1 from Q_1 to F_1. It remains now to prove that there exists an $\mathcal{R}$-rewriting Q_0 of Q_1 that maps to F_0. Either π_1 maps all atoms of Q_1 to F_0, and in that case $Q_0 = Q_1$ proves the property, or there is at least one atom a of Q_1 such that $\pi_1(a) \notin F_0$.

Since $F_1 = \alpha(F_0, R, \pi_\alpha)$, according to Lemma 8, there exists a unifier $\mu = (T_{F_1}, \pi_\alpha^{safe.1}(C), \pi_\alpha, id)$ of F_1 with $R = (H, C)$ such that $\beta(F_1, R, \mu)$ maps to F_0. Let us now build a unifier μ_1 of Q_1 with R, based upon μ and π_1, such that $Q_0 = \beta(Q_1, R, \mu_1)$ maps to F_0. This unifier $\mu_1 = (T_{Q_1}, Q'_1, \pi_\alpha, \pi'_1)$ is built as follows:

- T_{Q_1} is the subset of variables $t \in \mathit{var}(Q)$ such that $\exists t' \in \mathit{cutp}(R)$ with $\pi_\alpha(t') = \pi_1(t)$;
- Q'_1 is the subset of atoms from Q_1 such that $\pi_1(Q'_1) \subseteq {\pi_\alpha}^{safe.1}(C)$. Note that Q'_1 is a set of pieces according to T_{Q_1}, and that Q'_1 is not empty (there is an atom a such that $\pi_1(a) \notin F_0$);
- π'_1 is the restriction of π_1 to T_{Q_1}.

Let us now prove that there exists a homomorphism π_0 from $Q_0 = \beta(Q_1, R, \mu_1)$ to F_0. By definition, $Q_0 = {\pi_\alpha}^{safe.2}(H) \cup \pi'_1(Q_1 \setminus Q'_1)$. That homomorphism π_0 is built from the two following ones:

- the identity *id* is a homomorphism from $\pi'_1(Q_1 \setminus Q'_1)$ to F_0;
- π'_α maps $\pi_\alpha^{safe.2}(H)$ to F_0 and is defined as follows: $\forall t \in T_{Q_1}, \pi'_\alpha(t) = t$, otherwise $\pi'_\alpha(t) = \pi_\alpha(({\pi_\alpha}^{safe.2})^{-1}(t))$.

Since *id* and π'_α satisfy the condition of Property 23 (with $X = T_{Q_1}$), $\pi_0 = id + \pi'_\alpha$ is a homomorphism from Q_0 to F_0.

($\Leftarrow$) The second direction is proven by induction on the length of the rewriting sequence, and relies upon Lemma 7. The induction property is trivially true at rank 0. Let us assume it is true at rank n. Suppose that Q_{n+1} is obtained from Q by an $\mathcal{R}$-rewriting sequence $Q = Q_0, \ldots, Q_{n+1} = \beta(Q_n, R, \mu_\beta)$ of length $n+1$, and that there is a homomorphism π from Q_{n+1} to F. Since Q_n is obtained from Q by an $\mathcal{R}$-rewriting sequence of length n, and that Q_n obviously maps to itself, we know (by induction hypothesis) that there is an $\mathcal{R}$-derivation Q'_n of Q_n such that Q maps to Q'_n. According to Lemma 7, the fact $\alpha\beta(Q_n, R, \mu_\beta)$ has two properties: *(i)* it is obtained by

applying R on Q_{n+1}, and *(ii)* Q_n maps to $\alpha\beta(Q_n, R, \mu_\beta)$. From *(i)* and our knowledge that Q_{n+1} maps to F, we can conclude, thanks to Property 24, that there exists a fact F' obtained by the application of R to F such that $\alpha\beta(Q_n, R, \mu_\beta)$ (and thus Q_n, by composing homomorphisms) maps to F'. Since Q'_n is an $\mathcal{R}$-rewriting of Q_n, we can build, thanks to Corollary 1, a $\mathcal{R}$-rewriting F'' of F' such that Q'_n (and thus Q, by composition of homomorphisms), maps to F''. It remains now to point out that F'' is an $\mathcal{R}$-rewriting of F. □

C. Proof of Theorem 12

Theorem 12. ENTAILMENT *remains undecidable if* $\mathcal{R}$ *is composed of* fr1*-rules and* rr*-rules.*

Proof: We consider the halting problem of a Turing machine: given a deterministic Turing machine $\mathcal{M}$ (with an infinite tape and a single final state) and a word m, such that the head of $\mathcal{M}$ initially points to the first symbol of m, does $\mathcal{M}$ accept m, i.e., is there a sequence of transitions leading $\mathcal{M}$ to the final state? We build a reduction from this problem to ENTAILMENT, such that each rule obtained is *fr1* or *rr*. Let us call *configuration* a global state of the Turing machine, i.e., its current control state, the content of the tape and the position of the head. The basic idea of the translation is that each transition is translated into a logical rule. However, whereas transitions can be seen as *rewriting* rules, logical rules are only able to *add* atoms. To simulate the rewriting of a configuration, we add a library of eight rules, called hereafter the *copy rules*. The rule assigned to a transition creates three new cells (a copy of the current cell, that contains the new symbol, and neighboring cells with the new position of the head), and the copy rules build the other relevant cells at the right and at the left of these new cells.

Let $(\mathcal{M}, m)$ be an instance of the halting problem. We build an instance $(F, \mathcal{R}, Q)$ of ENTAILMENT as follows.

The vocabulary is composed of:

- binary predicates: *Succ* to encode the succession of cells (*Succ*(x, y) means that the cell x is followed by the cell y); *Value* to indicate the content of a given cell (*Value*(x, y) means that the cell x contains the symbol y); *Head* to indicate the current position of the head and the current control state (*Head*(x, y) means that the head points to cell x and the current state is y); *Next* to encode the rewriting of a cell (*Next*(x, y) means that cell x is rewritten as cell y); *Copy*$_r$ (resp. *Copy*$_l$) to rebuild the right (resp. the left) part of the word after a transition: *Copy*$_r(x, y)$ and *Copy*$_l(x, y)$ both mean that cell y is a copy of cell x in the next configuration;
- constants: each state and each symbol are translated into constants with the same name. Furthermore, there are three special constants, denoted □ (the value of an empty cell), B (for Begin) and E (for End).

Let $m = m_1 . \therefore . m_k$ and let T_0 be the initial state. F is obtained from this initial configuration. m is translated into a path of atoms with predicate *Succ* (a "*Succ*-path") on variables $x_1, \ldots, x_k$, as well as atoms with predicate *Value* that relate each x_i with the symbol m_i; for the needs of the copy mechanism, we actually translate the following

word: "$\Box\ m_1 \ldots m_k\ \Box$", and add special markers B and E at the extremities of this word. More precisely:
$F = \{$*Succ*(B, x_0),
Succ$(x_0, x_1), \ldots,$ *Succ*(x_k, x_{k+1}),
Succ(x_{k+1}, E),
Value$(x_0, \Box)$, *Value*$(x_{k+1}, \Box)$,
Value$(x_1, m_1), \ldots,$ *Value*(x_k, m_k),
Head$(x_1, T_0)\}$.

Note that there are no atoms of form *Value*$(B, \ldots)$ nor *Value*$(E, \ldots)$.

Let $\delta = (T_i, v_p) \rightarrow (T_j, v_q, d)$ be a transition, with $d \in \{r, l\}$ indicating a move to the right (r) or to the left (l): δ can be read as "if the current state is T_i and the head points to the symbol v_p, then take state T_j, replace v_p by v_q and move to the right/left". Let $R(\delta)$ be the logical rule assigned to δ. If $d = r$, we have:

$R(\delta)$ = *Head*$(x, T_i) \wedge$ *Value*$(x, v_p) \rightarrow$ *Next*$(x, y) \wedge$ *Succ*$(z, y) \wedge$ *Succ*$(y, t) \wedge$ *Value*$(y, v_q) \wedge$ *Head*(t, T_j). This rule is *fr1*. The case $d = l$ is symmetrical: the head moves to the left.

To implement the copy mechanism, we have four rules to copy the right part of the word, and four symmetrical rules to copy its left part. Here are the four "right-copy" rules:
$R_{r1} =$ *Succ*$(x, y) \wedge$ *Next*$(x, z) \wedge$ *Succ*$(z, u) \wedge$ *Value*$(y, v) \rightarrow$ *Copy*$_r(y, u) \wedge$ *Value*(u, v)
$R_{r2} =$ *Copy*$_r(x, y) \rightarrow$ *Succ*(y, z)
$R_{r3} =$ *Succ*$(x, y) \wedge$ *Copy*$_r(x, z) \wedge$ *Succ*$(z, u) \wedge$ *Value*$(y, v) \rightarrow$ *Copy*$_r(y, u) \wedge$ *Value*(u, v)
$R_{r4} =$ *Succ*$(x, E) \wedge$ *Copy*$_r(x, y) \wedge$ *Succ*$(y, z) \rightarrow$ *Value*$(z, \Box) \wedge$ *Succ*(z, E).

R_{r2} is *fr1* and the other rules are *rr* (with R_{r4} begin also *fr1*). In the "left-copy" rules, say $R_{l1} \ldots R_{l4}$, *Copy*$_l$ is used in an obvious way instead of *Copy*$_r$, with B replacing E. $\mathcal{R}$ contains these eight copy rules and one rule $R(\delta)$ per transition δ. Finally, Q encodes the fact that the head is in the final state: $Q = \{$*Head*$(x, T_f)\}$, where T_f is the final state.

The proof relies on the following equivalence: there is a derivation of F that contains a "*Succ*-path" from B to E encoding a word $\Box^* m' \Box^*$, with $Head(x, T)$ and $Value(x, m'_i)$, where x is a variable at a "position" k (with 0 being the position of the variable containing the beginning of m') *iff* there is a sequence of transitions of $\mathcal{M}$ from the initial configuration to a configuration with m' on the tape, the head pointing to a cell containing m'_i at a position k (with 0 being the position of the cell containing the beginning of m') and with control state T. The $\Rightarrow$ direction of this equivalence is proven by induction on the length of a derivation sequence. The $\Leftarrow$ direction is proven by induction on the number of transition applications.

□

Note (for Section 7.3). A similar encoding of the Turing Machine can be obtained in the following way. The initial word is encoded as before, but we add the atom *Next1*(x, x_1); note that x_0 and x_{k+1} are actually not needed, but we keep them to minimize the changes w.r.t. the previous encoding.

We consider two sets of rules. The first set is used to generate all the cells that could be needed by the Turing Machine (i.e., an infinite number of infinite tapes). These rules

are:

(R_1^1) $\mathit{Next1}(x, y) \rightarrow \mathit{Next1}(y, z) \wedge \mathit{Succ}(z', z)$

(R_1^2) $\mathit{Succ}(x, y) \rightarrow \mathit{Succ}(x', x)$

(R_1^3) $\mathit{Succ}(x, y) \rightarrow \mathit{Succ}(y, y')$

Note that these rules are both *fr1* and *ah*.

Intuitively, $\mathit{Next1}(x, x_1)$ in F is used by rule (R_1^1) as a starting point to generate an infinite number of tapes, while rules (R_1^2) and (R_1^3) generate an infinite number of cells for each tape. The second set of rules uses these cells to simulate a Turing Machine, without generating any new variable (these rules are *rr*). Note that cells B and E in the initial tape may be each linked by the *Next* predicate to an infinite path of empty cells (i.e., with value $\square$) in the second tape; for the following tapes, *Next* defines a bijection between cells of successive tapes.

(R_2^1) $\mathit{Next1}(x, y) \rightarrow \mathit{Next}(x, y)$

(R_2^{r2}) $\mathit{Next}(x, y) \wedge \mathit{Succ}(x, x') \wedge \mathit{Succ}(y, y') \rightarrow \mathit{Next}(x', y')$

(R_2^{l2}) $\mathit{Next}(x, y) \wedge \mathit{Succ}(x', x) \wedge \mathit{Succ}(y', y) \rightarrow \mathit{Next}(x', y')$

(R_2^{r3}) $\mathit{Next}(E, x) \wedge \mathit{Succ}(x, y) \rightarrow \mathit{Next}(E, y)$

(R_2^{l3}) $\mathit{Next}(B, x) \wedge \mathit{Succ}(y, x) \rightarrow \mathit{Next}(B, y)$

(R_2^{r4}) $\mathit{Next}(E, x) \rightarrow \mathit{Value}(x, \square)$

(R_2^{l4}) $\mathit{Next}(B, x) \rightarrow \mathit{Value}(x, \square)$

$R(\delta)$ [for the case $d = r$; replace $\mathit{Head}(y'', T_j)$ by $\mathit{Head}(y', T_j)$ for $d = l$]
$\mathit{Head}(x, T_i), Value(x, v_p), \mathit{Next}(x, y), \mathit{Succ}(x', x), \mathit{Succ}(x, x''), \mathit{Succ}(y', y), \mathit{Succ}(y, y'')$
$\rightarrow \mathit{Head}(y'', T_j), \mathit{Value}(y, v_q), \mathit{Copy}_l(x', y'), \mathit{Copy}_r(x'', y'')$

(R_2^{r5}) $\mathit{Copy}_r(x, y), \mathit{Value}(x, v), \mathit{Succ}(x, x'), \mathit{Succ}(y, y') \rightarrow \mathit{Copy}_r(x', y'), \mathit{Value}(y, v)$

(R_2^{l5}) $\mathit{Copy}_l(x, y), \mathit{Value}(x, v), \mathit{Succ}(x', x), \mathit{Succ}(y', y) \rightarrow \mathit{Copy}_l(x', y'), \mathit{Value}(y, v)$

The rules in the first set do not depend on the rules in the second set. This encoding of a Turing Machine can be seen either as an *ah*$\triangleright$*rr* or as a *fr1*$\triangleright$*rr* set of rules.

D. Proofs of Property 18 and Theorem 15

Property 18. *If a fact Q depends on a rule R, then there is a piece-unifier of Q with R.*

The proof of this property relies on the following definition and lemma, which states that the co-domain restriction on the substitution σ_R in the definition of a piece-unifier is only relevant for an algorithmic efficiency purpose.

We call *pre-unifier* of a fact Q with a rule $R = (H, C)$ a tuple $\mu = (T_Q, Q', \sigma_R, \pi_Q)$ where T_Q, Q' and π_Q are defined as in Definition 12, and σ_R is a substitution of $\mathit{fr}(R)$ by any set of terms (*without any restriction on its co-domain*).

Lemma 9. *[Pre-Unifier] Let Q be a fact and R be a rule. There exists a pre-unifier of Q with R if and only if there exists a piece-unifier of Q with R.*

Proof: Since every piece-unifier is a pre-unifier, we only have to prove the ($\Rightarrow$) part of this equivalence. Let us consider a pre-unifier $\mu = (T_Q, Q', \sigma_R, \pi_Q)$ of Q with $R = (H, C)$. We define $\mu' = (T_Q, Q', \sigma'_R, \pi'_Q)$ as follows:

- Let us first build the substitution σ'_R. To any term $t \in \sigma_R(\mathit{fr}(R))$ we associate a term $s(t) \in \mathit{cutp}(R) \cup \mathit{const}(Q')$, where s is an injective mapping obtained as follows:
  - if $t \in \mathit{cutp}(R) \cup \mathit{const}(Q')$, then $s(t) = t$;
  - otherwise, $s(t)$ is an element of $\mathit{fr}(R)$ (it is immediate to see that there will be enough such elements for s to be injective).

  Then we define $\sigma'_R = s \circ \sigma_R$.
- Now consider the substitution $\pi'_Q = s \circ \pi_Q$. It is effectively a homomorphism from Q' to $\sigma'_R(C)$ such that, for all $t \in T_Q$, there is a $t' \in \mathit{cutp}(R)$ with $\pi'_Q(t) = s(\pi_Q(t)) = s(\sigma_R(t')) = \sigma'_R(t')$.

Thus μ' is a piece-unifier of Q with R. □

Proof: (of Property 18) If Q depends on $R = (H, C)$, then there is a fact F and a homomorphism π from H to F such that there exists a homomorphism π' from Q to $F' = \alpha(F, R, \pi)$ that is not a homomorphism from Q to F (by Definition 25). In particular, it means that there is an atom a of Q such that $\pi'(a)$ is not in F. We consider $T_Q = \{x \in Q | \pi'(x) \in \pi(\mathit{cutp}(R))\}$. Consider now the non-empty subset Q' (it contains at least a) of atoms of Q that are not mapped to F by π', i.e., $Q' = \{a \in Q | \pi'(a) \notin F\}$. Check that Q' is a non-empty set of pieces of Q according to T_Q that contains a. Then the restriction π'' of π' to Q' is a homomorphism from Q' to $\pi^{\mathit{safe.1}}(C)$ such as, for every $t \in T_Q$, there is $t' \in \mathit{cutp}(R)$ with $\pi''(t) = \pi(t')$ (it immediately follows from the definition of T_Q). Thus $\mu = (T_Q, Q', \sigma_R, \pi'')$, where σ_R is the restriction of $\pi^{\mathit{safe.1}}$ to $\mathit{fr}(R)$, is a pre-unifier (not necessarily a unifier since π can map elements of $\mathit{fr}(R)$ to any term in F) of Q with R. >From Lemma 9 and its proof, it follows that $\mu_s = (T_Q, Q', s \circ \sigma_R, s \circ \pi'')$ is a unifier of Q with R. □

Theorem 15. *A fact Q depends on a rule $R = (H, C)$ if and only if there exists an atom-erasing unifier of Q with R.*

The following lemma will be used in the proof of the theorem.

Lemma 10. *Let $\mu = (T_Q, Q', \sigma_R, \pi_Q)$ be a unifier of Q with R. If there is a variable x of Q' such that $\pi_Q(x) \notin \sigma_R(\mathit{cutp}(R))$, then $\pi_Q(x)$ is not a variable of $\beta(Q, R, \mu)$.*

Proof: First see that x is not a variable of T_Q (or $\pi_Q(x)$ would be in $\sigma_R(\mathit{cutp}(R))$), and thus that x is not a variable of $Q \setminus Q'$ (otherwise the atom containing x would have been in a piece of Q'). Thus $\pi_Q(x)$ is not a variable of $\pi_Q(Q \setminus Q')$ (π_Q can only replace variables of T_Q by constants or by variables of $\sigma_R(\mathit{cutp}(R))$ that do not contain $\pi_Q(x)$, by hypothesis). Moreover, the terms of ${\sigma_R}^{\mathit{safe}}(H)$ only contain constants, variables of

$\sigma_R(\mathit{fr}(R))$, or safely substituted variables. Thus $\pi_Q(x)$ is not a variable of $\beta(Q, R, \mu)$. □

Proof: (of Theorem 15)

($\Rightarrow$) If Q depends on $R = (H, C)$, then consider the piece-unifier $\mu_s = (T_Q, Q', \sigma_s = s \circ \sigma_R, \pi_s = s \circ \pi'')$ of Q with R built in the proof of Property 18. The rewriting of Q according to R is thus the fact $Q_s = Q_s^1 \cup Q_s^2$ where $Q_s^1 = {\sigma_s}^{\mathit{safe}}(H)$ and $Q_s^2 = \pi_s(Q \setminus Q')$. Consider now the atom a of Q', such that $\pi'(a) \notin F$ as pointed out in the proof of Property 18. Either there is a variable x of a such that $\pi_s(x) \notin \sigma_s(\mathit{cutp}(R))$, or there is none.

- in the first case, $\pi_s(x)$ is not a variable of Q_s (see Lemma 10) and thus $\pi_s(a)$ is not an atom of Q_s;
- in the second case, we will show that $\pi_s(a)$ is neither an atom of Q_s^1, nor an atom of Q_s^2. Let us first suppose that $\pi_s(a)$ is in Q_s^1. That would mean that $\pi''(a)$ is an atom of F, and thus that $\pi'(a)$ is also an atom of F, which is absurd, due to our definition of a. Now suppose that $\pi_s(a)$ is in Q_s^2. It means that there is an atom $a' \in Q \setminus Q'$ such that $\pi_s(a) = \pi_s(a')$. Then, by construction of π_s, $\pi'(a) = \pi'(a')$. The atom $\pi'(a)$ is either in F or in $\pi^{\mathit{safe}}(C)$. The first case is absurd, according to our choice of a. The second is also absurd, since we would have chosen a' as an atom of Q'.

Since $\pi_s(a)$ is not an atom of Q_s, the unifier μ_s is atom-erasing.

($\Leftarrow$) If there is an atom-erasing unifier $\mu = (T_Q, Q', \sigma_R, \pi_Q)$ of Q with R, then consider the atom a in Q' such that $\pi_Q(a) \notin F = \beta(Q, R, \mu)$. Consider the natural homomorphism $\pi_1 + \pi_2$ from Q to $\alpha\beta(Q, R, \mu) = \alpha(F, R, {\sigma_R}^{\mathit{safe.1}}) = F \cup {\sigma_R}^{\mathit{safe.2}}(C)$ (see the proof of Lemma 7). It remains to prove that $(\pi_1 + \pi_2)(a) \notin F$. Only one of the following cases can arise:

- there is a variable x appearing in a such that $\pi_Q(x) \notin \sigma_R(\mathit{cutp}(R))$. In that case, $(\pi_1 + \pi_2)(x) = \pi_2(x) = {\sigma_R}^{\mathit{safe.2}}(\pi_Q(x))$ is a safely rewritten variable of ${\sigma_R}^{\mathit{safe.2}}(C)$ and cannot be in F, thus $(\pi_1 + \pi_2)(a) \notin F$.
- for every variable $x_i \in a, \pi_Q(x_i) \in \sigma_R(\mathit{cutp}(R))$. Then $(\pi_1 + \pi_2)(a) = \pi_2(a) = \pi_Q(a) \notin F$ since a is the erased atom.

□

References

[1] Abiteboul, S., Hull, R., Vianu, V., 1995. Foundations of Databases. Addison-Wesley.

[2] Aho, A. V., Beeri, C., Ullman, J. D., 1979. The theory of joins in relational databases. ACM Trans. Database Syst. 4 (3), 297–314.

[3] Baget, J.-F., Nov. 2001. Représenter des connaissances et raisonner avec des hypergraphes: de la projection à la dérivation sous contraintes. Ph.D. thesis, Université Montpellier II, France.

[4] Baget, J.-F., 2004. Improving the forward chaining algorithm for conceptual graphs rules. In: Proceedings of the Ninth International Conference on Principles of Knowledge Representation and Reasoning (KR2004), Whistler, Canada. AAAI Press, pp. 407–414.

[5] Baget, J.-F., Leclère, M., Mugnier, M.-L., 2010. Walking the decidability line for rules with existential variables. In: Proceedings of the Twelfth International Conference on Principles of Knowledge Representation and Reasoning, Toronto, Canada. AAAI Press, pp. 466–476.

[6] Baget, J.-F., Leclère, M., Mugnier, M.-L., Salvat, E., 2008. DL-$\mathcal{SR}$: a lite DL with expressive rules: Preliminary results. In: Proceedings of the Twenty-First International Workshop on Description Logics (DL2008), Dresden, Germany.

[7] Baget, J.-F., Leclère, M., Mugnier, M.-L., Salvat, E., 2009. Extending decidable cases for rules with existential variables. In: Proceedings of the Twenty-First International Joint Conference on Artificial Intelligence (IJCAI 2009), Pasadena, California, USA,. pp. 677–682.

[8] Baget, J.-F., Mugnier, M.-L., 2002. The Complexity of Rules and Constraints. J. Artif. Intell. Res. (JAIR) 16, 425–465.

[9] Baget, J.-F., Salvat, E., 2006. Rules dependencies in backward chaining of conceptual graphs rules. In: ICCS. Vol. 4068 of LNCS. Springer, pp. 102–116.

[10] Beeri, C., Vardi, M., 1981. The implication problem for data dependencies. In: Proceedings of Automata, Languages and Programming, Eigth Colloquium (ICALP 1981), Acre (Akko), Israel. Vol. 115 of LNCS. pp. 73–85.

[11] Beeri, C., Vardi, M., 1984. A proof procedure for data dependencies. Journal of the ACM 31 (4), 718–741.

[12] Calì, A., Gottlob, G., Kifer, M., 2008. Taming the infinite chase: Query answering under expressive relational constraints. In: Proceedings of the Eleventh International Conference on the Principles of Knowledge Representation and Reasoning (KR 2008), Sydney, Australia. pp. 70–80.

[13] Calì, A., Gottlob, G., Lukasiewicz, T., 2009. A general datalog-based framework for tractable query answering over ontologies. In: Proceedings of the Twenty-Eigth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems (PODS 2009), Providence, Rhode Island, USA. pp. 77–86.

[14] Calì, A., Gottlob, G., Lukasiewicz, T., 2010. Datalog extensions for tractable query answering over ontologies. In: Virgilio, R. D., Giunchiglia, F., Tanca, L. (Eds.), Semantic Web Information Management: A Model-Based Perspective. Springer, pp. 249–279.

[15] Calì, A., Gottlob, G., Lukasiewicz, T., Marnette, B., Pieris, A., 2010. Datalog+/-: A family of logical knowledge representation and query languages for new applications. In: LICS. IEEE Computer Society, pp. 228–242.

[16] Calì, A., Kifer, M., 2006. Containment of conjunctive object meta-queries. In: Proceedings of the Thirty-Second International Conference on Very Large Data Bases (VLDB 2006), Seoul, Korea. pp. 942–952.

[17] Calì, A., Lembo, D., Rosati, R., 2003. On the decidability and complexity of query answering over inconsistent and incomplete databases. In: Proceedings of the Twenty-Second ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS 2003), San Diego, California, USA. ACM, pp. 260–271.

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

[19] Chandra, A. K., Lewis, H. R., Makowsky, J. A., 1981. Embedded implicational dependencies and their inference problem. In: Proceedings of the Thirteenth Annual ACM Symposium on Theory of Computing (STOC 1981), Milwaukee, Wisconsin, USA. ACM, pp. 342–354.

[20] Chandra, A. K., Merlin, P. M., 1977. Optimal implementation of conjunctive queries in relational data bases. In: STOC. pp. 77–90.

[21] Chandra, A. K., Vardi, M. Y., 1985. The implication problem for functional and inclusion dependencies is undecidable. SIAM J. Comput. 14 (3), 671–677.

[22] Chein, M., Mugnier, M.-L., 1992. Conceptual Graphs: Fundamental Notions. Revue d'Intelligence Artificielle 6 (4), 365–406.

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

[24] Coulondre, S., 2003. A top-down proof procedure for generalized data dependencies. Acta Inf. 39 (1), 1–29.

[25] Courcelle, B., 1990. The monadic second-order logic of graphs: I. recognizable sets of finite graphs. Inf. Comput. 85 (1), 12–75.

[26] Deutsch, A., Nash, A., Remmel, J., 2008. The chase revisited. In: Twenty-Eigth ACM SIGMOD/PODS International Conference on Management of Data / Principles of Database Systems (PODS 2008), Vancouver, BC, Canada. pp. 149–158.

[27] Deutsch, A., Nash, A., Remmel, J. B., 2008. The chase revisited. In: PODS. pp. 149–158.

[28] Deutsch, A., Tannen, V., 2003. Reformulation of xml queries and constraints. In: Ninth International Conference on Database Theory (ICDT 2003), Siena, Italy. pp. 225–241.

[29] Fagin, R., Kolaitis, P. G., Miller, R. J., Popa, L., 2003. Data exchange: Semantics and query answering. In: Ninth International Conference on Database Theory (PODS 2003), Siena, Italy. pp. 207–224.

[30] Fagin, R., Kolaitis, P. G., Miller, R. J., Popa, L., 2005. Data exchange: semantics and query answering. Theor. Comput. Sci. 336 (1), 89–124.

[31] Hayes, P. (Ed.), 2004. RDF Semantics. W3C Recommendation. W3C, http://www.w3.org/TR/rdf-mt/.

[32] Johnson, D., Klug, A., 1984. Testing containment of conjunctive queries under functional and inclusion dependencies. J. Comput. Syst. Sci. 28 (1), 167–189.

[33] Lutz, C., Toman, D., Wolter, F., 2009. Conjunctive query answering in the description logic el using a relational database system. In: Proceedings of the Twenty-First International Joint Conference on Artificial Intelligence (IJCAI 2009), Pasadena, California, USA,. pp. 2070–2075.

[34] Maier, D., Mendelzon, A. O., Sagiv, Y., 1979. Testing implications of data dependencies. ACM Trans. Database Syst. 4 (4), 455–469.

[35] Salvat, E., Mugnier, M.-L., 1996. Sound and Complete Forward and Backward Chainings of Graph Rules. In: Proceedings of the Fourth International Conference on Conceptual Structures (ICCS'96), Sydney, Australia. Vol. 1115 of LNAI. Springer, pp. 248–262.

[36] Sowa, J. F., 1984. Conceptual Structures: Information Processing in Mind and Machine. Addison-Wesley.

[37] Thomas, R., 1988. The tree-width compactness theorem for hypergraphs, http://people.math.gatech.edu/∼thomas/PAP/twcpt.pdf.

[38] Vardi, M., 1984. The implication and finite implication problems for typed template dependencies. J. Comput. Syst. Sci. 28 (1), 3–28.