

Ontological Query Answering with Existential Rules

Marie-Laure Mugnier

University Montpellier 2, France

Abstract. The need for an ontological layer on top of data, associated with advanced reasoning mechanisms able to exploit the semantics encoded in ontologies, has been acknowledged both in the database and knowledge representation communities. We focus in this paper on the ontological query answering problem, which consists of querying data while taking ontological knowledge into account. To tackle this problem, we consider a logical framework based on existential rules, also called Tuple-Generating Dependencies or Datalog \pm rules. This framework can also be defined in graph terms. Query entailment with existential rules is not decidable, thus a crucial issue is to define decidable classes of rules as large as possible. This paper is a survey of decidable classes of rules, including a review of the main complexity results. It mostly relies on previous work presented at IJCAI'2009 [BLMS09] and KR'2010 [BLM10] (and developed in a journal paper [BLMS11]), updated to include very recent results.

1 Introduction

In this paper, we consider rules that have the ability of generating new unknown individuals, an ability called value invention in databases [AHV95]. These rules are of the form $body \rightarrow head$, where the body and the head are conjunctions of atoms (without functions) and variables that occur only in the head are *existentially* quantified, hence their name $\forall\exists$ -rules in [BLMS09, BLM10] or existential rules in [BMRT11, KR11].

Example 1. Consider the existential rule $R = \forall x(Human(x) \rightarrow \exists y(isParent(y, x) \wedge Human(y)))$ and a fact $F = Human(A)$, where A is a constant. The application of R to F produces new factual knowledge, namely $\exists y_0(isParent(y_0, A) \wedge Human(y_0))$, where y_0 is a variable denoting an unknown individual. Note that R could be applied again to $Human(y_0)$, which would lead to create another existentially quantified variable, and so on.

Existential rules are known in databases as Tuple-Generating Dependencies (TGDs) [BV84]. TGDs have been extensively used as a high-level generalization of different kinds of constraints, e.g., for data exchange [FKMP05]. They also correspond to rules in conceptual graphs, a graph-based knowledge representation formalism [Sow84] [CM09]. Recently, there has been renewed interest for these rules in the context of ontological query answering, a topical problem both in knowledge representation and in databases.

Given the complexity and ever increasing amounts of data nowadays available, the need for an ontological layer on top of data, associated with advanced querying mechanisms able to exploit the semantics encoded in ontologies, has been acknowledged

both in the database and knowledge representation communities. In this paper, we will reserve the term “ontology” to general domain knowledge—sometimes also called terminological knowledge—in order to clearly distinguish it from the data—or assertional knowledge—called here facts. Given a knowledge base (KB) composed of an ontology and of facts, and a query, the ontological query problem consists in computing the set of answers to the query on the KB, while taking implicit knowledge represented in the ontology into account. This problem is also known as ontology-based data access. E.g., on Example 1: the Boolean query $\exists x \text{ isParent}(x, A)$ (“does A have a parent ?”) has a negative answer on F alone, but a positive answer when the knowledge encoded in the rule R is taken into account. Queries are supposed to be at least as expressive as (Boolean) conjunctive queries in databases, which can be seen as existentially closed conjunctions of atoms.

In the Semantic Web, ontological knowledge is often represented with formalisms based on description logics (DLs). However, DLs traditionally focused on reasoning tasks about the ontology itself (the so-called TBox), for instance classifying concepts; querying tasks were restricted to ground atom entailment. Conjunctive query answering with classical DLs has appeared to be extremely complex (e.g., for the classical DL \mathcal{ALCC} , it is 2EXPTIME -complete, and still NP-complete in the size of the data). Hence, less expressive DLs specially devoted to conjunctive query answering on large amounts of data have been designed recently, namely DL-Lite [CGL⁺07], \mathcal{EL} [BBL05,LTW09], and their generalization to Horn-logics (see e.g., [KRH07]). These DLs are the basis of the so-called tractable profiles of the Semantic Web language OWL 2.

On the other hand, querying large amounts of data is the fundamental task of databases. Therefore, the challenge in this domain is now to access data while taking ontological knowledge into account. The deductive database language Datalog allows to express some ontological knowledge. However, in Datalog rules, variables are range-restricted, i.e., all variables in the rule head necessarily occur in the rule body, which does not allow for value invention. This feature has been recognized as crucial in an open-world perspective, where it cannot be assumed that all individuals are known in advance. This motivated the recent extension of Datalog to TGDs (i.e., existential rules), which gave rise to the Datalog +/- family [CGK08,CGL09,CGL⁺10b].

Existential rules have some particularly interesting features in the context of the Web. On the one hand, they cover the core of lightweight DLs dedicated to query answering, while being more powerful and flexible [CGL09,BLM10,BMRT11]. In particular, they have unrestricted predicate arity (while DLs consider unary and binary predicates only), which allows for a natural coupling with database schemas, in which relations may have any arity; moreover, adding pieces of information, for instance to take contextual knowledge into account, is made easy by the unrestricted predicate arity, since these pieces can be added as new predicate arguments. On the other hand, existential rules cover Datalog, while allowing for value invention.

Let us mention that our own work on existential rules is related to our earlier studies on conceptual graphs [CM09]. Indeed, the logical translation of conceptual graph rules yields exactly existential rules [SM96]. Inspired by conceptual graphs, we have developed a knowledge representation framework, which can be seen both as logic-based and graph-based, i.e., the knowledge constructs can be also be seen as graphs, with a logical

translation, and reasoning mechanisms are based on graph-theoretic notions, while being sound and complete with respect to entailment in the associated logical fragments [CM09].

In the following, we will focus on the fundamental decision problem associated with query answering based on existential rules, namely Boolean conjunctive query answering, or conjunctive query entailment: given a knowledge base K composed of a set of existential rules and facts and a (Boolean) conjunctive query Q , does K give rise to an answer to Q , i.e., is Q entailed by K (noted $K \models Q$)? The ability to generate existential variables, associated with arbitrarily complex conjunctions of atoms, makes this problem undecidable in general [BV81,CLM81]. Since the birth of TGDs, and recently within the Datalog+/- and existential rule frameworks, various conditions of decidability have been exhibited.

This paper is a survey of decidable classes of rules, including a review of the main complexity results in terms of combined and data complexities. An important issue is whether these decidable classes can be combined while keeping decidability. We recall that the rough union of decidable sets of rules almost always leads to undecidability, and present a tool, the graph of rule dependencies, which allows to define new decidability conditions by constraining possible interactions between rules.

This survey mostly relies on previous papers presented at IJCAI'2009 [BLMS09] and KR'2010 [BLM10] (and developed in a journal paper [BLMS11]), updated to include very recent results [CGP10a,CGP10b,KR11,BMRT11].

Section 2 provides basic notions on existential rules. Section 3 outlines the associated graph-based framework. Section 4 reports decidability and complexity results. Section 5 is devoted to combining decidable paradigms via the graph of rule dependencies. We conclude with some open issues.

2 Preliminaries

We consider first-order logical languages with constants but no other function symbols. A *term* is thus a variable or a constant. An *atom* is of the form $p(t_1, \dots, t_k)$ where p is a predicate with arity k , and the t_i are terms. A *ground* atom contains only constants. A *conjunct* $C[\mathbf{x}]$ is a finite conjunction of atoms, where \mathbf{x} is the set of variables occurring in C . A *fact* is the existential closure of a conjunct.¹ A (Boolean) *conjunctive query* (CQ) has the same form as a fact, thus we identify both notions. We also see conjuncts, facts and CQs as sets of atoms. Given an atom or a set of atoms A , $\text{vars}(A)$, $\text{consts}(A)$ and $\text{terms}(A)$ denote its set of variables, of constants and of terms, respectively. First-order semantic entailment is denoted by \models and semantic equivalence by \equiv .

Given conjuncts F and Q , a *homomorphism* π from Q to F is a substitution of $\text{vars}(Q)$ by $\text{terms}(F)$ such that $\pi(Q) \subseteq F$ (we say that Q maps to F by π ; Q and F are

¹ In the literature, a fact is traditionally a ground atom. Since existential rules produce atoms with variables that are existentially quantified, we generalize the notion of a fact to an existentially closed conjunction of atoms. Moreover, this allows to cover naturally languages such as RDF/S, in which a blank node is logically translated into an existentially quantified variable, or basic conceptual graphs.

respectively called the source and the target of the homomorphism). It is well-known that, given two facts F and Q , $F \models Q$ iff there is a homomorphism from Q to F .

Definition 1 ($\forall\exists$ -Rule). A $\forall\exists$ -rule (existential rule, or simply rule when not ambiguous) is a formula $R = \forall x\forall y(B[x, y] \rightarrow (\exists zH[y, z]))$ where $B = \text{body}(R)$ and $H = \text{head}(R)$ are conjuncts, resp. called the body and the head of R . The frontier of R , noted $\text{fr}(R)$, is the set of variables $\text{vars}(B) \cap \text{vars}(H) = \mathbf{y}$.

In the following, we will omit quantifiers in rules as there is no ambiguity.

Note that an existential 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) an existential rule can be transformed into a set of Horn clauses with functions.

Definition 2 (Application of a Rule). A rule R is applicable to a fact F if there is a homomorphism π from $\text{body}(R)$ to F ; the result of the application of R on F w.r.t. π is a fact $\alpha(F, R, \pi) = F \cup \pi^{\text{safe}}(\text{head}(R))$ where π^{safe} is a substitution of $\text{head}(R)$, that replaces each $x \in \text{fr}(R)$ with $\pi(x)$, and each other variable with a “fresh” variable not introduced before; this application is said to be redundant if $\alpha(F, R, \pi) \equiv F$.

Example 2. Consider the following predicates, with their arity mentioned in parentheses; unary predicates can be seen as concept names, i.e. types of entities, and the other predicates as relation names: $\text{Area}(1)$, $\text{Project}(1)$, $\text{Researcher}(1)$, $\text{isProject}(3)$, $\text{hasExpertise}(2)$, $\text{isMember}(2)$

Here are some examples of rules composing the ontology:

“The relation isProject associates a project, the area of this project and the leader of this project, who is a researcher”

$R_0 = \text{isProject}(x, y, z) \rightarrow \text{Project}(x) \wedge \text{Area}(y) \wedge \text{Researcher}(z)$ [signature of isProject]

“Every leader of a project is a member of this project”

$R_1 = \text{isProject}(x, y, z) \rightarrow \text{isMember}(z, x)$

“Every researcher expert in an area is member of a project in this area”

$R_2 = \text{Researcher}(x) \wedge \text{hasExpertise}(x, y) \rightarrow \text{isProject}(p, y, z) \wedge \text{isMember}(x, p)$

Let $F = \{\text{Researcher}(A), \text{hasExpertise}(A, KR), \text{Area}(KR)\}$ be a fact. R_2 is applicable to F , which yields $F' = F \cup \{\text{isProject}(p_1, KR, z_1), \text{isMember}(A, p_1)\}$.

Definition 3 (Derivation Sequence). Let F be a fact, and \mathcal{R} be a set of rules. An \mathcal{R} -derivation of F is a finite sequence $(F_0 = F), \dots, F_k$ s.t. for all $0 \leq i < k$, there is $R_i \in \mathcal{R}$ and a homomorphism π_i from $\text{body}(R_i)$ to F_i s.t. $F_{i+1} = \alpha(F_i, R_i, \pi_i)$.

Theorem 1 (Completeness of Forward Chaining). Let F and Q be two facts, and \mathcal{R} be a set of rules. Then $F, \mathcal{R} \models Q$ iff there exists an \mathcal{R} -derivation $(F_0 = F), \dots, F_k$ such that $F_k \models Q$.

It follows that a breadth-first forward chaining mechanism yields a positive answer in finite time when $F, \mathcal{R} \models Q$. This mechanism, called the *saturation* hereafter (and the chase in databases) works as follows. Let F_0 be the initial fact F . Each step consists

of checking if Q maps to the current fact, say F_{i-1} at step i ($i \geq 1$), and otherwise producing a fact F_i from F_{i-1} , by computing all new homomorphisms from each rule body to F_{i-1} , then performing all corresponding rule applications. A homomorphism is said to be new if it has not been already computed at a previous step, i.e., it uses at least an atom added at step $i - 1$ ($i \geq 2$). The fact F_k obtained after the step k is called the k -saturation of F and is denoted by $\alpha_k(F, \mathcal{R})$.

A *knowledge base* (KB) $\mathcal{K} = (F, \mathcal{R})$ is composed of a finite set of facts (seen as a single fact) F and a finite set of rules \mathcal{R} . The (*Boolean*) *CQ entailment* problem (denoted ENTAILMENT hereafter) is the following: given a KB $\mathcal{K} = (F, \mathcal{R})$ and a (*Boolean*) CQ Q , does $F, \mathcal{R} \models Q$ hold?

This framework can be extended to *equality rules* and *constraints*. An equality rule is a rule of the form $B \rightarrow x = t$, where x and t are distinct terms, $x \in \text{vars}(B)$ and $t \in \text{vars}(B)$ or is a constant. When the unique name assumption is made, i.e., distinct constants refer to distinct individuals, the application of an equality rule is said to fail if it leads to set the equality between distinct constants. This kind of failure corresponds to an inconsistency of the knowledge base. Equality rules generalize functional dependencies, which are widely used in data modeling and ontologies. Constraints are another kind of construct specifically devoted to the definition of the consistency or inconsistency of the knowledge base. A *negative* constraint is a rule of the form $C \rightarrow \perp$, where \perp denotes the absurd symbol (i.e., a propositional atom whose value is false), or equivalently $\neg C$. It is satisfied if C is not entailed by (F, \mathcal{R}) . A *positive* constraint has the same form as an existential rule. It is satisfied if every homomorphism from its body to a fact F' entailed by (F, \mathcal{R}) , where \mathcal{R} may include equality rules, is extendable to a homomorphism from its head to F' . Negative constraints are typically used to express disjointness of concepts/classes or incompatibility of relations, while positive constraints require some pieces of knowledge to be present or entailed (cf. the classical use of TGDs in databases). See [CGL09] and [BLMS11] for the integration of equality rules and negative constraints in the existential rule framework. See [BM02] for a framework including existential rules (without equality) and both negative and positive constraints. In this paper, we will only consider existential rules.

The two classical ways of processing rules are forward chaining, introduced above, and *backward chaining*. Instead of using rules to enrich the facts, the backward chaining proceeds in the “reverse” manner: it uses the rules to *rewrite* the query in different ways with the aim of producing a query that maps to the facts. The key operation in this mechanism is the *unification* operation between part of a current goal (a conjunctive query or a fact in our framework) and a rule head. This mechanism is typically used in logic programming, with rules having a head restricted to a single atom, which is unified with an atom of the current goal. Since the head of an existential 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 heads and goals without decomposing them into single atoms, using a graph notion called a *piece* (see Section 5.2). The operator that rewrites the query is denoted by β and is informally defined as follows (for a formal definition see [BLMS09][BLMS11]): given a conjunct Q , a rule $R = B \rightarrow H$ and a piece-unifier μ of (part of) Q with (the head of)

R , $\beta(Q, R, \mu) = Q_\mu \cup B_\mu$ where Q_μ is a specialization of the non-unified subset of Q (determined by μ), and B_μ is a specialization of the body of R (also determined by μ).

Definition 4 (Rewriting sequence). *Let Q be a conjunct and \mathcal{R} be a set of rules. An \mathcal{R} -rewriting of Q is a finite sequence $(Q_0 = Q), Q_1, \dots, Q_k$ s. t. for all $0 \leq i < k$, there is $R_i \in \mathcal{R}$ and a piece-unifier μ of Q_i with (the head of) R_i such that $Q_{i+1} = \beta(Q_i, R_i, \mu)$.*

The soundness and completeness of the backward chaining mechanism can be proven via the following equivalence with the forward chaining: there is an \mathcal{R} -rewriting from the query Q to a query Q' that maps to the initial fact F iff there is an \mathcal{R} -derivation from F to a fact F' such that Q maps to F' .

3 Graphical View of the Framework

Although we present our framework in a logical setting, it is also graph-based. Indeed, we also view facts, queries, rules (and constraints) as labeled graphs or hypergraphs; entailment between facts/queries is computed by a graph or hypergraph homomorphism, which corresponds to the homomorphism notion defined on formulas; entailment using rules (and constraints) relies on homomorphism in the same way as in the logical framework. In this section, we will first briefly present this graphical framework.

Generally speaking, seeing formulas as graphs or hypergraphs allows to focus on their structure: notions like paths, cycles, tree decompositions are then natural. For instance, in this paper, the *bts* decidable class of rules or the *piece* notion used to define unifiers both rely on a graph view of the formulas. Moreover, the graph setting allows to benefit from techniques and results in graph theory, and other areas such as constraint programming (indeed, there are straightforward reductions between the problem of the existence of a homomorphism between two labeled graphs/hypergraphs and the problem of the consistency of a constraint network).²

A fact, or simply a set of atoms, F can be naturally seen as an ordered labeled hypergraph, whose nodes and hyperedges respectively encode the terms and the atoms from F . More precisely, in the hypergraph $\mathcal{F} = (X, \mathcal{E}, l)$ assigned to F , X is a set of nodes in bijection with $terms(F)$, \mathcal{E} is a multiset³ of hyperedges, which are tuples on X , in bijection with the set of atoms in F , and l is a labeling function of nodes and edges. A node is labeled by c if the corresponding term in F is a constant c (otherwise, the term is not labeled, or, equivalently, labeled by a “blank” label). A hyperedge corresponding to an atom $p(t_1, \dots, t_k)$ in F is labeled by p and is equal to the tuple $(v_{t_1}, \dots, v_{t_k})$, where v_{t_i} is the node assigned to t_i .

One may also consider F as a graph, which is exactly the *incidence graph* of \mathcal{F} (called here a “basic graph”): it is a bipartite undirected multigraph (i.e., there may be several edges between two nodes), with one set of nodes representing the terms (i.e., the nodes in \mathcal{F}), called term nodes, and the other set of nodes representing the atoms (i.e., the hyperedges in \mathcal{F}), called relation nodes. For each atom $p(t_1, \dots, t_k)$ in F , instead

² The constraints are supposed to be defined in extension, which corresponds to the basic “constraint satisfaction problem (CSP)”.

³ In a multiset, the same element may appear several times.

of a hyperedge, there is a relation node labeled by p and this node is incident to k edges linking it to the nodes assigned to t_1, \dots, 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 [CM09].

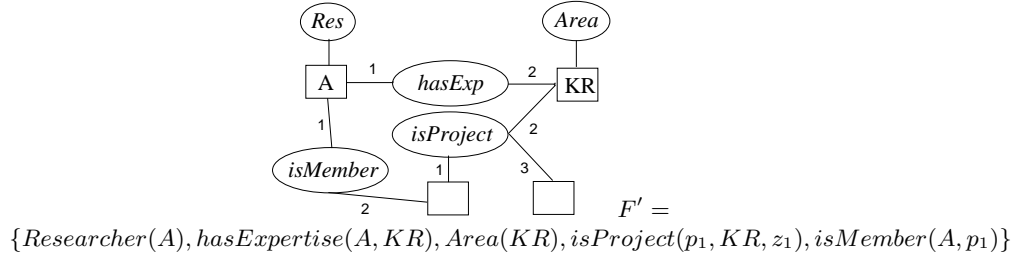


Fig. 1. Basic graph corresponding to F' (Example 2)

The bipartite graph view is more suitable for drawing, while the hypergraph view is often more efficient from an algorithmic viewpoint (as it yields a global view of an atom as a hyperedge).

Generally speaking, a homomorphism maps a relational structure to another relational structure, while preserving the information encoded in the first structure. When the structures are sets of atoms, the homomorphism maps terms to terms, while preserving the information encoded in the terms (the constants here) and atoms. When the structures are labeled hypergraphs (resp. graphs), it maps nodes to nodes while preserving the information encoded in the labels and hyperedges (resp. edges). Thus, a homomorphism π from a hypergraph $\mathcal{H} = (X, \mathcal{E}, l)$ to a hypergraph $\mathcal{H}' = (X', \mathcal{E}', l')$ is a mapping from X to X' such that: (1) for each node $v \in X$, $l(v) = l(\pi(v))$ if $l(v)$ is a constant; (2) for each hyperedge $e = (v_1 \dots v_k) \in \mathcal{E}$, there is a hyperedge $e' = (\pi(v_1) \dots \pi(v_k)) \in \mathcal{E}'$ such that $l(e) = l'(e')$. If we consider bipartite graphs instead of hypergraphs, a homomorphism π from G to G' is a mapping such that: (1) the node bipartition is preserved, i.e., π maps term nodes from G to term nodes from G' and relation nodes from G to relation nodes from G' ; (2) for each term node v in G labeled by a constant and for each relation node in G , $\pi(v)$ has the same label as v ; (3) for each edge (r, t) labeled by i in G , $(\pi(r), \pi(t))$ is an edge labeled by i in G' . Condition (3) can be equivalently expressed as follows: for each relation node a in G with list of neighbors $(v_1 \dots v_k)$ —where v_i is the extremity of the i th edge incident to a — $\pi(a)$ has list of neighbors $(\pi(v_1) \dots \pi(v_k))$.

It is immediately checked that there is a one-to-one correspondence between homomorphisms of two formulas and homomorphisms of their corresponding hypergraphs. Moreover, there is a one-to-one correspondence between homomorphisms of two hypergraphs and homomorphisms of their corresponding bipartite graphs if we consider only term nodes (otherwise, if duplicate atoms are allowed in the target conjunct F' , there may be several graph homomorphisms for a single hypergraph homomorphism).

A rule can be seen as a bicolored basic graph (and similarly as a bicolored hypergraph), with the first color for the body and the second color for the other elements; then, the head is composed of the subgraph induced by nodes of the second color plus the nodes corresponding to the frontier of the rule. See Figure 2, where the body of the rule is colored in white and the other nodes in gray. Rule application and other homomorphism-based notions can be translated in a straightforward way.

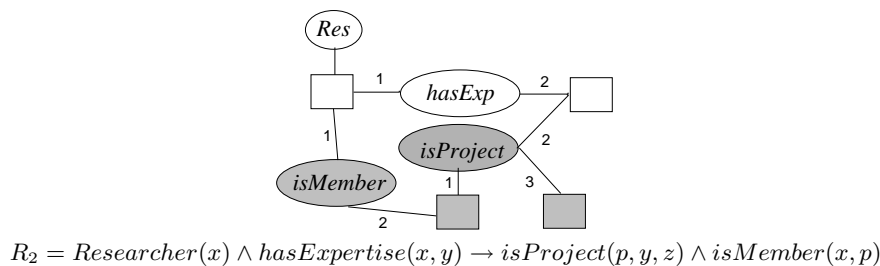


Fig. 2. Graph rule corresponding to R_2 (Example 2)

4 The Landscape of Decidable Classes

The **ENTAILMENT** problem is known to be non-decidable ([BV81,CLM81] on TGDs), even if the set of rules is restricted to a single rule [BLM10].

4.1 Abstract Characterizations

Decidable classes found in the literature are based on various syntactic properties of existential rules. In order to classify them, three abstract properties related to the behavior of reasoning mechanisms are considered in [BLMS09][BLM10]: the forward chaining halts in finite time; the forward chaining may not halt but the facts generated have a tree-like structure; the backward chaining mechanism halts in finite time. These properties yield three *abstract* classes of rules, respectively called *finite expansion sets*, *bounded treewidth sets* and *finite unification sets*. These classes are said to be abstract in the sense that they do not come with a syntactic property that can be checked on rules or sets of rules. As a matter of fact, none of these classes is recognizable, i.e., the problem of determining whether a given set of rules fulfills the abstract property is not decidable [BLM10].

We first specify these three notions. A set of rules \mathcal{R} is said to be a *finite expansion set* (*fes*) if, for every fact F , there exists an integer k such that $\alpha_k(F, \mathcal{R}) \equiv \alpha_{k+1}(F, \mathcal{R})$, i.e., all rule applications to $\alpha_k(F, \mathcal{R})$ are redundant [BM02]. Weaker versions, in the sense that they allow to stop in less cases, can be considered. For instance the halting condition may be $\alpha_k(F, \mathcal{R}) = \alpha_{k+1}(F, \mathcal{R})$, i.e., no new rule application can be performed on $\alpha_k(F, \mathcal{R})$; the saturation algorithm with this halting condition corresponds

to the so-called *oblivious* chase in databases (note that the chase variant called the *restricted* chase is still weaker than *fes*). If \mathcal{R} is a *fes*, then the termination is guaranteed for any forward chaining that (1) builds a derivation sequence until the halting condition is satisfied (the order in which rules are applied does no matter), then (2) checks if the query maps to the obtained fact.

Bounded-treewidth sets of rules form a more general class, which was essentially introduced in [CGK08]. The following definition of the *treewidth* of a fact corresponds to the usual definition of the treewidth of a graph, where the considered graph is the primal graph of the hypergraph of the fact (this graph has the same set of nodes as the hypergraph and there is an edge between two nodes if they belong to the same hyperedge).

Definition 5 (Treewidth of a fact). Let F be a fact. A tree decomposition of F is an undirected tree T with set of nodes $\mathcal{X} = \{X_1, \dots, X_k\}$ where:

1. $\bigcup_i X_i = \text{terms}(F)$;
2. for each atom a in F , there is $X_i \in \mathcal{X}$ s.t. $\text{terms}(a) \subseteq X_i$;
3. for each term e in F , the subgraph of T induced by the nodes X_i that contain e is connected (“running intersection property”).

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

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 (for instance with the saturation algorithm), $\text{treewidth}(F') \leq b$. A *fes* is a *bts*, since the finite saturated graph generated by a *fes* has a treewidth bounded by its size.

Proving the decidability of ENTAILMENT with *bts* is not as immediate as with *fes*. Indeed, the proof relies on a theorem from Courcelle [Cou90], that states that classes of first-order logic having the bounded treewidth model property are decidable. This proof does not (at least not directly) provide a halting algorithm.

Very recently, a subclass of *bts* has been defined, namely *greedy bts (gbts)*, which is equipped with a halting algorithm [BMRT11]. This class is defined as follows. A derivation is said to be *greedy* if, for every rule application in this derivation, all the frontier variables not being mapped to the initial fact are jointly mapped to terms added by a single previous rule application. This allows to build a tree decomposition of a derived fact in a greedy way: let T_0 be the set of terms occurring in the initial fact F and of all constants occurring in the rules; the root of the tree, X_0 , is equal to T_0 ; all other nodes in the tree will contain T_0 as well; the *i*th rule application of a rule R with homomorphism π leads to create a node $X_i = \pi^{\text{safe}}(\text{head}(R)) \cup T_0$ and an edge between X_i and the node X_j such that j is the smallest integer for which $\pi^{\text{safe}}(\text{fr}(R)) \subseteq \text{terms}(X_j)$ (since the derivation is greedy, there is such X_j). This yields a tree decomposition of width bounded by $|T_0| \cup \max(\text{vars}(\text{head}(R)))_{R \in \mathcal{R}}$.

The third class, *finite unification set (fus)* [BLMS09], requires that the number of rewritings of Q using the rules is finite for any fact. More precisely, one considers only the “most general” rewritings of Q , the other rewritings being useless for the querying task. 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 . A set of rules \mathcal{R} is called a *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' . Note that it may be the case that the set of the most general rewritings is finite while the set of rewritings is infinite.

If \mathcal{R} is a *fus*, then a backward chaining algorithm that builds rewritings of Q in a breadth-first way, while maintaining a set \mathcal{Q} of the most general \mathcal{R} -rewritings built, and answers yes if an element of \mathcal{Q} maps to F , necessarily halts in finite time.

The *fes* and *fus* classes are not comparable, neither are *bts* (resp. *gbts*) and *fus*.

4.2 Concrete Decidable Classes

Let us now enumerate the main concrete classes. Most of them implement one of the three preceding abstract behaviors; however, some concrete classes that are not *bts* neither *fus* have been exhibited very recently [CGP10a], we will mention them in this section.

The typical *fes* concrete class is plain Datalog, where rules do not have any existential variable in their head, i.e., for any Datalog rule R , $\text{vars}(\text{head}(R)) \subseteq \text{vars}(\text{body}(R))$. Other names for this class are *range-restricted rules* (*rr*) [AHV95], *full* implicational dependencies [CLM81] and *total* tuple-generating dependencies [BV84]. These rules typically allow to express specialization relationships between concepts or relations in ontological languages, as well as properties of relations such as reflexivity, symmetry or transitivity.

A special class is that of *disconnected* (*disc*) rules, which have an empty frontier [BM02]. A disconnected rule needs to be applied only once: any further application of it is redundant; this is why these rules are both *fes* and *fus*. Moreover, *disc*-rules have the nice property of being compatible with any other decidable class (see Section 5). The body and the head of a *disc*-rule may share constants, which allows to express knowledge about specific individuals. Apart from this use, this class is mostly useful in technical constructions.

Other *fes* cases are obtained by restricting possible interactions between rules. These interactions have been encoded in two different directed graphs: a graph encoding variable sharing between positions in predicates and a graph encoding dependencies between rules. In the first graph, called (*position*) *dependency graph* [FKMP03] [FKMP05], the nodes represent positions in predicates, i.e., the node (p, i) represents a position i in predicate p . Then, for each rule R and each variable x in $\text{body}(R)$ occurring in position (p, i) , edges with origin (p, i) are built as follows: if $x \in \text{fr}(R)$, there is an edge from (p, i) to each position of x in $\text{head}(R)$; furthermore, for each existential variable y in $\text{head}(R)$ (i.e., $y \in \text{vars}(\text{head}(R)) \setminus \text{fr}(R)$) occurring in position (q, j) , there is a special edge from (p, i) to (q, j) . A set of rules is said to be *weakly acyclic* (*wa*) if its position dependency graph has no circuit passing through a special edge. Intuitively, such a circuit means that the introduction of an existential variable in a given position may lead to create another existential variable in the same position, hence an infinite number of existential variables. The weak-acyclicity property is a sufficient condition (but of course not a necessary condition) for the forward chaining to be finite [FKMP03][DT03]. Recently, weak-acyclicity has been independently generalized

in various ways, namely *safety* [MSL09], *super-weak-acyclicity* [Mar09] and *joint-acyclicity* [KR11], while keeping the forward chaining finiteness property. Note that *joint-acyclicity* (*ja*) is obtained by simply shifting the focus from positions to existential variables, hence replacing the position dependency graph by the *existential dependency graph*, where the nodes are the existential variables occurring in rules; this yields a finer analysis of potentially infinite creations of existential variables.

In the second graph, called *graph of rule dependencies* (*GRD*), the nodes represent rules and the edges represent dependencies between rules. The GRD is precisely defined in Section 5.2. The acyclicity of the GRD, noted *aGRD* in Figure 4.2, ensures that the forward chaining, as well as the backward chaining, is finite, thus *aGRD* is both a *fes* and a *fus* class (see Section 5.2). More generally, when all strongly connected components of the GRD have the property of being weakly-acyclic sets of rules (noted *wa-GRD*), then the forward chaining is finite (special case of Theorem 4 in Section 5.2); this class corresponds to the notion of a stratified chase graph in [DNR08].

Let us now review *gbts* classes, which, intuitively, ensure that the derived facts have a tree-like structure that can be built in a greedy way.

The notion of a guarded rule is inspired from guarded logic [AvBN96]. A rule R is *guarded* (g) if there is an atom a in its body (called a guard) that contains all variables from the body, i.e., $\text{vars}(\text{body}(R)) \subseteq \text{vars}(a)$. A generalization of guarded rules is obtained by relaxing the guardedness property: a set of rules is *weakly guarded* (wg) if, for each rule R , there is $a \in \text{body}(R)$ (called a weak guard) that contains all *affected* variables from $\text{body}(R)$. The notion of an affected variable is relative to the rule set: a variable is affected if it occurs only in affected predicate positions, which are positions that may contain a new variable generated by forward chaining (see [FKMP05] for a precise definition). The important property is that a rule application necessarily maps non-affected variables to terms from the initial fact.

A rule R is *frontier-one* ($fr1$) if its frontier is of size one (note that rules restricted to a frontier of size two still lead to undecidability). By noticing that the shape of derived facts depends only on how the frontier of rules is mapped (and not on how the whole body is mapped, since only the images of the frontier are used to apply a rule), one obtains a generalization of both *fr1*- and guarded-rules: a rule R is *frontier-guarded* (fg) if there is an atom a in its body that contains all variables in its frontier, i.e., $\text{vars}(\text{fr}(R)) \subseteq \text{vars}(a)$. The same remark as for guarded rules can be made: only affected variables need to be guarded. One then obtains a generalization of both wg and fg : a set of rules is *weakly-frontier-guarded* (wfg) if, for each rule R , there is $a \in \text{body}(R)$ that contains all affected variables from $\text{fr}(R)$. In a very recent paper [KR11], the class $w(f)g$ is further generalized into *jointly-(frontier)-guarded* ($(j-f)g$), by refining the notion of affected variable.

Interestingly, [KR11] exhibits a class that is *bts* but neither *fes* nor *gbts*, namely *glut-frontier-guarded* ($glut-fg$). This class generalizes both notions of joint-acyclicity (which itself generalizes weak-acyclicity) and joint-(frontier)-guardedness: a set of rules is *glut-(frontier)-guarded* if each rule has an atom in its body that contains all *glut* variables (occurring in its frontier). This class relies on a special method for eliminating existential quantifiers; instead of being replaced by functional terms as in skolemisation, existential quantifiers are replaced by “flattened” functional terms encoded as ad-

ditional arguments in predicates. Briefly, the *glut* variables are the variables that remain affected after this rule rewriting.

Whether the *gbts* class is concrete, i.e. recognizable, is not known yet. Note that guarded rules and *wg*-rules were already provided with an algorithm [CGK08][CGL09], but that it was not the case for *frl*-rules and their generalizations up to $(j\text{-}(f)g)$ -rules, which can now benefit from the algorithm for *gbts*. A *glut-fg* set of rules can be translated into an exponentially large *j-fg* set of rules, thus the *glut-fg* class is also provided with an algorithm.

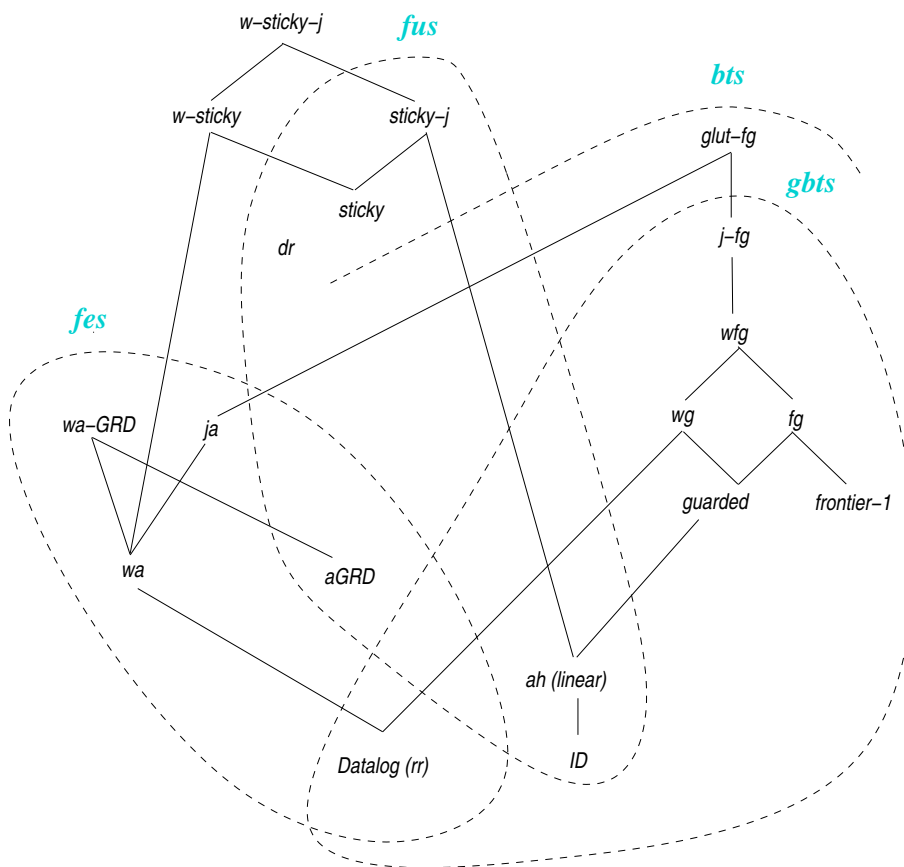
About *fus* concrete cases, two classes are exhibited in [BLMS09]. The first class is that of *atomic-hypothesis rules* (*ah*)—where “hypothesis” stands for “body”—whose body is restricted to a single atom; these rules are also called linear TGDs [AHV95]. Since *ah*-rules are *fus*, there is a halting algorithm based on backward chaining, but, since they are also special guarded rules, there is also a halting algorithm based on forward chaining. Atomic-hypothesis rules are useful to express necessary properties of concepts or relations in ontological languages, without any restriction on the form of the head, i.e., by rules of the form $C(x) \rightarrow P$ or $r(x_1, \dots, x_k) \rightarrow P$, where C is a concept, r a k -ary relation and P any set of atoms. Specific *ah*-rules translate the so-called *inclusion dependencies* (*ID*) in databases: the body and the head of these rules are each composed of a single atom, whose arguments are pairwise distinct variables.

The second class of rules, *domain-restricted rules* (*dr*), constrains the form of the head: each atom in the head contains all or none of the variables in the body. For instance, a domain-restricted rule can express the so-called *concept-product*, argued to be a useful constructor for description logics in [RKH08]: this operator allows to compute the cartesian product of two concepts by rules of the form $p(x) \wedge q(y) \rightarrow r(x, y)$ (e.g., $elephant(x) \wedge mouse(y) \rightarrow bigger\text{-}than(x, y)$).

In [CGP10a], another concrete *fus* class is defined: *sticky* rules, which are incomparable with *ah*-rules and *dr*-rules. The stickyness property restricts multiple occurrences of variables (in the same atom or in distinct atoms—i.e., in joins) in the rule bodies. Variables that occur in rule bodies are marked according to the following procedure: (1) for each rule R , for each variable x in $body(R)$, if there is an atom in $head(R)$ that does not contain x , then mark every occurrence of x in $body(R)$; (2) repeat the following step until a fixpoint is reached: for each rule R , if a marked variable in $body(R)$ appears at position (p, i) then, for every rule R' (including $R = R'$) and every variable x appearing in position (p, i) in $head(R')$, mark every occurrence of x in $body(R')$. A set of rules \mathcal{R} is said to be *sticky* if there is no rule $R \in \mathcal{R}$ such that a marked variable occurs in $body(R)$ more than once. The above mentioned concept-product rule is obviously sticky since no variable occurs twice in the rule body.

Several generalizations of sticky rules are defined in [CGP10b]. All these classes are obtained by more sophisticated variable-marking techniques. *Weakly-sticky* (*w-sticky*) sets are a generalization of both weakly-acyclic sets and sticky sets: intuitively, if a marked variable occurs more than once in a rule body, then at least one of these positions has to be safe, i.e., only a finite number of terms can appear in this position during the forward chaining. *Sticky join* (*sticky-j*) sets generalize sticky sets. Finally, *weakly-sticky-join* (*w-sticky-j*) sets generalize both weakly-sticky sets and sticky-join sets. These classes are still incomparable with *dr*.

Figure 4.2 synthesizes inclusions between the preceding concrete decidable classes. All inclusions are strict and classes not related in the schema are indeed incomparable. Each class belongs to at least one of the abstract classes *fes*, *fus*, *gbts* and *bts*, except for the two recent classes weakly-sticky and weakly-sticky-join: indeed, they generalize both a *fes* but not *fus* nor *gbts* class, namely *wa*, and a *fus* but not *bts* class, namely *sticky*.



Moreover, *disc* is included in *wa*, *dr* and *fg*

Fig. 3. Inclusions between decidable cases

4.3 Complexity

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, thus the sizes of \mathcal{R} and Q can be seen as bounded by constants. 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. An intermediate notion of complexity is found in the literature, namely *knowledge base* complexity: in this case, not only the data is considered as input, but the whole knowledge base, i.e., \mathcal{K} in our framework. However, we can translate Q into a rule $R_Q = Q \rightarrow match$ where *match* is a fresh nullary predicate, with the entailment question becoming $F, \mathcal{R} \cup \{R_Q\} \models match$. Thus, knowledge base complexity is often less relevant in our framework: each time this translation can be done while keeping the wanted property of the initial rule set, knowledge base complexity and combined complexity coincide.

Table 4.3 summarizes the combined and data complexity results for the main concrete classes mentioned in Section 4.2. Note that combined complexity is here without bound of the predicate arity (putting an upper-bound on the arity of predicates may decrease the complexity). By definition, all *fus* classes have polynomial data complexity, since the number of rewritten queries is not related to the data size. They are even *first-order rewritable*, which means that every query Q can be rewritten as a first-order query Q' using the set of rules, such that the evaluation of Q on the initial KB (\mathcal{R}, F) produces the same set of answers as the evaluation of Q' on F . An interest of first-order queries is that they can be encoded in SQL, which allows to use relational database systems, thus benefiting from their optimizations. Obviously, any Boolean query over a *fus* class can be rewritten as a first-order query, which is the union (i.e., disjunction) of all most general queries in the set \mathcal{Q} . It is well-known in databases that deciding whether a first-order query is entailed by a database belongs to the class AC^0 in data complexity (AC^0 is a subclass of LSpace –for logarithmic space– itself included in PTime). Several non-*fus* classes have polynomial data complexity: some *gbts* classes, namely *fg* (and its subclasses *fr1* and *guarded*), some *fes* classes, namely *wa-GRD* and *ja* (and subclasses *aGRD*, *wa* and *Datalog*) and some non-*bts* classes, namely *w-sticky-j* (and its subclass *w-sticky*). Note that relaxing guardedness into weak-guardedness leads to EXPTIME-complete data complexity.

5 Combining Decidable Classes

In the previous section, we have reviewed the main concrete classes of rules found in the literature. These rules rely on different criteria ensuring decidability and sometimes tractability in data complexity. The question now is whether these criteria can be combined to make larger decidable classes. The answer to this question is also of interest if we want to use jointly two ontologies, possibly provided by an alignment also expressed as a set of rules; assume that each ontology is known to correspond to a decidable class, as well as the alignment; the question is whether these ontologies can be safely combined. In this section, we will first present negative results showing that the rough union of two classes is almost never decidable, then introduce a technique allowing to combine decidable classes and decidability paradigms under some conditions.

| Class | Combined Complexity | Data Complexity |
|-----------------------|--------------------------------------|-----------------------------------|
| gbts | in 3EXPTIME [BMRT11] (1) | EXPTIME-c [BMRT11] |
| glut-fg | 3EXPTIME-c [KR11] | EXPTIME-hard |
| j-fg | 2EXPTIME-c [KR11] | EXPTIME-c [KR11] |
| wfg | 2EXPTIME-c [BMRT11] | EXPTIME-c [BMRT11] |
| fg | 2EXPTIME-c [BMRT11] | PTIME-c [BMRT11] |
| fr1 | 2EXPTIME-c [BMRT11] | PTIME-c [BMRT11] |
| wg | 2EXPTIME-c [CGK08] | EXPTIME-c [CGL10a] |
| guarded | 2EXPTIME-c [CGK08] | PTIME-c [CGL09] |
| Datalog (rr) | EXPTIME-c e.g., [CLM81] | PTIME-c [DEGV01] |
| j-a | 2EXPTIME-c [KR11] | PTIME-c [KR11] |
| wa, wa-GRD (2) | 2EXPTIME-c [CGP10b](LB) [FKMP05](UB) | PTIME-c [DEGV01](LB) [FKMP05](UB) |
| linear (ah) | PSPACE-c [CGL10a] | FO-rewritable [CGL09] |
| sticky | EXPTIME-c [CGP10a] | FO-rewritable [CGP10a] |
| sticky-j | EXPTIME-c [CGP10b] | FO-rewritable [CGP10b] |
| w-sticky | 2EXPTIME-c [CGP10b] | PTIME-c [CGP10b] |
| w-sticky-j | 2EXPTIME-c [CGP10b] | PTIME-c [CGP10b] |

(1) 2EXPTIME-completeness is proven in an extended yet unpublished version of [BMRT11]

(2) These complexities have been proven for *wa*, but hold also for *wa-GRD*

Table 1. Combined and Data Complexities for the main concrete decidable classes

Let us mention the specific case of disconnected rules, which are universally compatible: if a set of rules \mathcal{R} is decidable, then the union of \mathcal{R} and any set of disconnected rules remains decidable [BLM10].

5.1 Rough Union

We say that two sets of rules \mathcal{R}_1 and \mathcal{R}_2 are *equivalent* w.r.t. a vocabulary \mathcal{V} composed of a set of predicates and a set of constants, 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. We consider here two simple transformations from a rule into an equivalent pair of rules namely τ_1 and τ_2 :

- τ_1 rewrites a rule R into two rules:
 $R_1 = \text{body}(R) \rightarrow R(x_1 \dots x_p)$ and
 $R_2 = R(x_1 \dots x_p) \rightarrow \text{head}(R)$, where $\{x_1, \dots, x_p\} = \text{vars}(\text{body}(R))$ and R is a new predicate (i.e., not belonging to the vocabulary) assigned to R . Note that R_1 is both *rr* (plain Datalog) and *dr*, and R_2 is *ah* (linear TGD).
- τ_2 is similar to τ_1 , except that the atom with predicate R contains all variables in the rule R :
 $R_1 = \text{body}(R) \rightarrow R(y_1, \dots, y_k)$ and
 $R_2 = R(y_1, \dots, y_k) \rightarrow \text{head}(R)$, where $\{y_1, \dots, y_k\} = \text{vars}(R)$. Note that, among other properties, R_1 is *dr*, while R_2 is *rr*.

Any set of rules can be split into an equivalent set of rules by τ_1 or τ_2 . If we furthermore consider the concrete classes of the rules obtained by both transformations, and knowing that ENTAILMENT is undecidable with a single rule, we obtain the following result:

Theorem 2. [BLM10] ENTAILMENT remains undecidable if \mathcal{R} is composed of

- a range-restricted (plain Datalog) rule and an atomic-hypothesis rule
- a range-restricted (plain Datalog) 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 incompatibility of *frl* and *rr* can be proven with a reduction from the halting problem of a Turing Machine [BLM10][BLMS11]. The compatibility of *frl* and *dr* is an open question. The possible compatibility of *sticky* with other classes has not been studied yet.

Among decidability criteria, it is important to distinguish between properties that can be checked on each rule (“individual” properties) and properties to be checked on the set of rules (“global” properties) like weak-guardedness, weak-acyclicity, GRD acyclicity or stickyness. Indeed, the union of two sets satisfying an individual property still satisfies it, while this is not true for global properties: a single added rule may lead to violate any of the global properties mentioned in this paper. For instance, a *wa* set of rules is weakly-sticky, and the same holds for a sticky set of rules, but the union of two such sets is generally not weakly-sticky. The concrete classes *wg*, *wfg*, *wa* and *aGRD*, all based on global properties, are pairwise incompatible, which includes the incompatibility of each class with itself [BLM10]. It follows from previous results that abstract classes are incompatible: the union of two sets belonging to classes *fes*, *bts* (*gbts*) or *fus* does not preserve decidability.

In summary, the rough union of two sets of rules belonging to different decidable classes almost always leads to undecidability. The next question is whether decidable sets can be combined under some constraints. The following section introduces the “graph of rule dependencies” and define conditions on the structure of this graph that preserve decidability.

5.2 Rule Dependencies

Intuitively, we say that a rule R_2 depends on a rule R_1 if R_1 may bring knowledge that leads to a new application of R_2 . More formally: there exists a fact F such that R_1 is applicable to F but R_2 is not, and there is an application of R_1 to F leading to F' such that R_2 is applicable to F' .

The Graph of Rule Dependencies (*GRD*) encodes dependencies between rules from a set \mathcal{R} . It is a directed graph with \mathcal{R} as the set of nodes and an edge (R_i, R_j) if R_j depends on R_i .⁴

⁴ For historical reasons, the edges encode the converse of the dependency relation (an edge (R_i, R_j) can be read as “ R_i may lead to trigger R_j in a new way”).

This abstract dependency relation can be effectively computed with a *unification* operation. As already mentioned, this unification operation takes the complex structure of rule heads into account, that is why it is not simply a unification between two atoms. Indeed, whereas in Datalog, it is possible to decompose a rule $B \rightarrow A_1 \dots A_k$ into an equivalent set of k rules of the form $(B \rightarrow A_i)_{1 \leq i \leq k}$, with atomic heads, such a transformation would not preserve the rule semantics when applied to existential rules. Instead, one has to consider the so-called *pieces* in the head, which can be seen as “units” of knowledge brought by an application of the rule.

Pieces in a rule head are defined as follows. Generally speaking, a piece of a set of atoms A according to a subset of $\text{vars}(A)$, denoted by X , is a minimal non-empty subset P of A such that, for all a and a' in A , if $a \in P$ and $(\text{vars}(a) \cap \text{vars}(a')) \not\subseteq X$, then $a' \in P$. In our case, A is the head of the rule and X is its frontier. With a graphical view of a rule head, pieces can be recast as follows: two atom nodes a_1 and a_2 are in the same piece if there is a *path* between them that goes through existential variable nodes only.

Example 3. [Pieces] Cf. Figure 4.

$R = p(x, y) \rightarrow p(x, z) \wedge p(z, t) \wedge p(t, x) \wedge p(x, u) \wedge p(u, x)$. The frontier of R is $\{x\}$, hence R has two pieces $\{p(x, z), p(z, t), p(t, x)\}$ and $\{p(x, u), p(u, x)\}$.

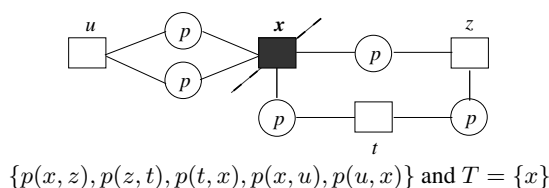


Fig. 4. Pieces

Then, any rule $R = B \rightarrow P_1 \dots P_k$, where the P_i are the pieces in R can be recast as an equivalent set of k rules of the form $(B \rightarrow P_i)_{1 \leq i \leq k}$.

The backward chaining defined in [BLMS09] (and previously in [SM96] for conceptual graphs) relies on unifiers based on pieces, called piece-unifiers. Briefly said, a piece-unifier of the body of R_2 with the head of R_1 is a homomorphism μ from a subset of $\text{body}(R_2)$ to a specialization H'_1 of $\text{head}(R_1)$ [with $H'_1 = s(\text{head}(R_1))$, where s is a substitution of $\text{fr}(R_1)$ with $\text{fr}(R_1) \cup \text{consts}(\text{head}(R_1) \cup \text{body}(R_2))$] that satisfies the following condition: let T_2 be the set of variables from $\text{body}(R_2)$ mapped by μ to $\text{fr}(R_1) \cup \text{consts}(\text{head}(R_1))$; consider the pieces of $\text{body}(R_2)$ according to T_2 ; then, μ is a homomorphism from some pieces of $\text{body}(R_2)$ to H'_1 . See [BLMS09][BLMS11] for formal definitions.

Piece-unifiers allow to effectively compute dependencies between rules: R_2 depends on R_1 iff there is a piece-unifier between $\text{body}(R_2)$ and $\text{head}(R_1)$, that satisfies a simple syntactic condition (see [BLMS11] for details):

Theorem 3. [BLMS11] R_2 depends on R_1 if and only if there exists an atom-erasing piece-unifier of $\text{body}(R_2)$ with $\text{head}(R_1)$.

Example 4. Let $\mathcal{R}_1 = \{R_1, R_2\}$, with:

$$R_1 = p(x) \rightarrow r(x, y) \wedge r(y, z) \wedge r(z, x)$$

$$R_2 = r(x, y) \wedge r(y, x) \rightarrow p(x)$$

R_1 depends on R_2 , but R_2 does not depend on R_1 . Indeed, let us see $\text{head}(R_1)$ and $\text{body}(R_2)$ as graphs; in this example, any piece-unifier of $\text{body}(R_2)$ with $\text{head}(R_1)$ is necessarily a homomorphism from $\text{body}(R_2)$ to $\text{head}(R_1)$. Since the cycle in $\text{body}(R_2)$ does not map by homomorphism to the cycle in $\text{head}(R_1)$, R_2 does not depend on R_1 .

The associated decision problem (given two rules R_1 and R_2 , does R_2 depend on R_1 ?) is NP-complete. The GRD notion has been first introduced for conceptual graph rules in [Bag04], then adapted to existential rules in [BLMS09]; the notion of a piece-unifier defined in [BLMS09] is itself adapted from a similar notion defined for backward chaining with conceptual graph rules [SM96]. A notion equivalent to the GRD, called the chase graph, has been independently defined for TGDs in [DNR08].

Let us consider the basic saturation mechanism. 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 $\{R' \in \mathcal{R} \mid \exists R \in S, (R, R') \text{ is an edge in } \text{GRD}(\mathcal{R})\}$. Similar arguments apply for backward chaining, considering the predecessors of the rules instead of their successors. It follows that for any set of rules \mathcal{R} , if $\text{GRD}(\mathcal{R})$ has no circuit, then \mathcal{R} is both a *fes* and a *fus*. This result can be extended by considering the strongly connected components of $\text{GRD}(\mathcal{R})$. 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.

Theorem 4. [BLMS09] Let \mathcal{R} be a set of rules. If all strongly connected components of $\text{GRD}(\mathcal{R})$ are *fes* (resp. *fus*), then \mathcal{R} is a *fes* (resp. *fus*).

In [DNR08], it is proven that when all strongly connected components of the GRD (called the chase graph) are weakly-acyclic (the chase graph is said to be stratified) the forward-chaining is finite, which can be seen as a special case of the previous result.

The above results allow to safely combine several *fes*, or several *fus*. We will now combine *fes/bts* and *fus*.

Definition 6 (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.

Theorem 5. [BLMS09] [BLMS11] 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 $P, \mathcal{R}_2 \models Q$ hold.

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.

Theorem 6 (*fes* \triangleright *bts*). [BLMS09] [BLMS11] *The class fes* \triangleright *bts* *is a subclass of bts.*

The following result allows to combine both forward and backward chaining mechanisms, provided that a specific kind of cut exists.

Theorem 7 (*bts* \triangleright *fus*). [BLMS09] [BLMS11] *ENTAILMENT is decidable when restricted to the bts* \triangleright *fus* *class of rules.*

In the specific case of a *fes* \triangleright *fus* set provided with an appropriate cut $\mathcal{R}_1 \triangleright \mathcal{R}_2$, 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' , and 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' .

6 Conclusion

In this paper, we have presented a rule-based framework well-suited to ontological query answering, which can be seen both logically and graphically. These rules allow for value invention, which has been recognized as a mandatory feature of ontological languages in an open-world perspective. We have given an overview of the landscape of decidable classes of rules in relationship with classical computational paradigms, namely forward chaining and backward chaining, and reviewed the main complexity results, for combined and data complexity. The rough union of decidable rule sets is generally not decidable, but conditions on the interaction between rules defined on the graph of rule dependencies allow for safe union, as well as for combining the forward and backward chaining mechanisms.

The study of existential rules for ontological query answering is only at its beginning and a lot of issues are to be addressed. We list below some challenging problems directly related to this paper.

- *Extend or define new abstract decidable classes.* Abstract decidable classes are useful to highlight the properties of reasoning mechanisms (i.e., forward and backward chaining in this paper) that ensure decidability. However, some very recent concrete classes do not belong to any of the abstract classes *fes*, *fus* and *bts*. Is it possible to cover them by combining abstract classes and properties of the GRD, or do they correspond to a new decidability paradigm? Another question is the following: the *fes* and *fus* classes, both based on the finiteness of rule processing mechanisms can be seen as playing a similar role, with respect to forward chaining and backward chaining respectively. Is there an abstract class that would be the counterpart of *bts* for backward chaining?

- *Find safe ways of integrating restricted forms of transitivity and/or equality.* Two kinds of rules, specially useful for modeling applications, are well-known sources of undecidability when they interact with existential rules: transitivity rules (and more generally rules allowing to compose binary relations) and equality rules. For instance, it has long been shown in databases that functional dependencies (which are specific equality rules) and inclusion dependencies (which are specific *ah* rules) make entailment undecidable [CV85]; moreover, although equality rules can be safely added to plain Datalog, this is not the case for *fes* classes in general: adding a single equality rule to a *fes* may lead to undecidability [BLMS11]. Current techniques for safely integrating equality rules enforce a “separability” condition between equality rules and existential rules so that they can be processed separately (see in particular [CGL09]): intuitively, this condition ensures that equality rules can then be considered as a set of constraints to be satisfied by the initial facts, and query entailment considers existential rules only. It would be nice to have decidable cases allowing some interaction between equality rules and existential rules during the forward chaining process.
- *Deepen the analysis of interactions between rules.* We have pointed out the interest of precisely studying interactions between rules to extend decidable cases. The notion of rule dependencies could be refined. A first step in that direction is the generalization of rule dependencies to rule *k*-dependencies, which allows to take into account several steps of saturation instead of a single one [BMT11]. Another technique for analyzing interactions between rules is the graph of position dependencies, mentioned in Section 4.2, which leads to the notion of weak acyclicity. These two techniques encode different kinds of interactions between rules, hence define incomparable decidable classes, and this remains true for their currently known generalizations, except for *wa-GRD* which combines *wa* with a condition on the GRD. However, this way of doing does not really “integrate” both notions.
- *Optimize and evaluate algorithms.* Polynomial data complexity is a requirement, however it does not ensure practical feasibility. In particular, any backward chaining algorithm has “by definition” a polynomial data complexity on a *fus* class, however the number of generated queries can be prohibitively large in practice. There is still much work to do to go from algorithmic schemes to scalable algorithms. Among other techniques, the graph of rule dependencies can be seen as a compilation technique to improve the efficiency of the forward and backward chaining mechanisms, thus speeding up the query answering task.

Acknowledgements. The author thanks Sebastian Rudolph for his careful reading of this paper and helpful comments.

References

- AHV95. S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases*. Addison-Wesley, 1995.
- AvBN96. H. Andréka, J. van Benthem, and I. Németi. Modal languages and bounded fragments of FOL. Research Report ML-96-03, Univ. of Amsterdam, 1996.

- Bag04. J.-F. Baget. Improving the forward chaining algorithm for conceptual graphs rules. In *KR'04*, pages 407–414. AAAI Press, 2004.
- BBL05. F. Baader, S. Brandt, and C. Lutz. Pushing the el envelope. In *IJCAI'05*, pages 364–369, 2005.
- BLM10. J.-F. Baget, M. Leclère, and M.-L. Mugnier. Walking the decidability line for rules with existential variables. In *KR'10*, pages 466–476. AAAI Press, 2010.
- BLMS09. J.-F. Baget, M. Leclère, M.-L. Mugnier, and E. Salvat. Extending decidable cases for rules with existential variables. In *IJCAI'09*, pages 677–682, 2009.
- BLMS11. J.-F. Baget, M. Leclère, M.-L. Mugnier, and E. Salvat. On rules with existential variables: Walking the decidability line. *Artificial Intelligence*, 175(9-10):1620–1654, 2011.
- BM02. J.-F. Baget and M.-L. Mugnier. The Complexity of Rules and Constraints. *J. Artif. Intell. Res. (JAIR)*, 16:425–465, 2002.
- BMRT11. J.-F. Baget, M.-L. Mugnier, S. Rudolph, and M. Thomazo. Walking the complexity lines for generalized guarded existential rules. In *IJCAI'11, to appear*, 2011.
- BMT11. J.F. Baget, M.-L. Mugnier, and M. Thomazo. Towards farsighted dependencies for existential rules. Research report lirmm 11-016, 2011.
- BV81. C. Beeri and M. Vardi. The implication problem for data dependencies. In *ICALP'81*, volume 115 of *LNCS*, pages 73–85, 1981.
- BV84. C. Beeri and M.Y. Vardi. A proof procedure for data dependencies. *Journal of the ACM*, 31(4):718–741, 1984.
- CGK08. A. Cali, G. Gottlob, and M. Kifer. Taming the infinite chase: Query answering under expressive relational constraints. In *KR'08*, pages 70–80, 2008.
- CGL⁺07. D. Calvanese, G. De Giacomo, D. Lembo, M. Lenzerini, and R. Rosati. Tractable reasoning and efficient query answering in description logics: The DL-Lite family. *J. Autom. Reasoning*, 39(3):385–429, 2007.
- CGL09. A. Cali, G. Gottlob, and T. Lukasiewicz. A general datalog-based framework for tractable query answering over ontologies. In *PODS'09*, pages 77–86, 2009.
- CGL10a. A. Cali, G. Gottlob, and T. Lukasiewicz. Datalog extensions for tractable query answering over ontologies. In R. De Virgilio, F. Giunchiglia, and L. Tanca, editors, *Semantic Web Information Management: A Model-Based Perspective*, pages 249–279. Springer, 2010.
- CGL⁺10b. A. Cali, G. Gottlob, T. Lukasiewicz, B. Marnette, and A. Pieris. Datalog+/-: A family of logical knowledge representation and query languages for new applications. In *LICS*, pages 228–242. IEEE Computer Society, 2010.
- CGP10a. A. Cali, G. Gottlob, and A. Pieris. Advanced processing for ontological queries. *PVLDB*, 3(1):554–565, 2010.
- CGP10b. A. Cali, G. Gottlob, and A. Pieris. Query answering under non-guarded rules in datalog+/- . In *RR*, pages 1–17, 2010.
- CLM81. A. K. Chandra, H. R. Lewis, and J. A. Makowsky. Embedded implicational dependencies and their inference problem. In *STOC'81*, pages 342–354. ACM, 1981.
- CM09. M. Chein and M.-L. Mugnier. *Graph-based Knowledge Representation and Reasoning—Computational Foundations of Conceptual Graphs*. Advanced Information and Knowledge Processing. Springer, 2009.
- Cou90. B. Courcelle. The monadic second-order logic of graphs: I. recognizable sets of finite graphs. *Inf. Comput.*, 85(1):12–75, 1990.
- CV85. A. K. Chandra and M. Y. Vardi. The implication problem for functional and inclusion dependencies is undecidable. *SIAM J. Comput.*, 14(3):671–677, 1985.
- DEGV01. E. Dantsin, T. Eiter, G. Gottlob, and A. Voronkov. Complexity and expressive power of logic programming. *ACM Comput. Surv.*, 33(3):374–425, 2001.

- DNR08. A. Deutsch, A. Nash, and J.B. Remmel. The chase revisited. In *PODS'08*, pages 149–158, 2008.
- DT03. A. Deutsch and V. Tannen. Reformulation of xml queries and constraints. In *ICDT'03*, pages 225–241, 2003.
- FKMP03. R. Fagin, P. G. Kolaitis, R. J. Miller, and L. Popa. Data exchange: Semantics and query answering. In *ICDT'03*, pages 207–224, 2003.
- FKMP05. R. Fagin, P. G. Kolaitis, R. J. Miller, and L. Popa. Data exchange: semantics and query answering. *Theor. Comput. Sci.*, 336(1):89–124, 2005.
- KR11. M. Krötzsch and S. Rudolph. Extending decidable existential rules by joining acyclicity and guardedness. In *IJCAI'11, to appear*, 2011.
- KRH07. M. Krötzsch, S. Rudolph, and P. Hitzler. Complexity boundaries for Horn description logics. In *AAAI'07*, pages 452–457. AAAI Press, 2007.
- LTW09. C. Lutz, D. Toman, and F. Wolter. Conjunctive query answering in the description logic el using a relational database system. In *IJCAI'09*, pages 2070–2075, 2009.
- Mar09. B. Marnette. Generalized schema-mappings: from termination to tractability. In *PODS*, pages 13–22, 2009.
- MSL09. M. Meier, M. Schmidt, and G. Lausen. On chase termination beyond stratification. *PVLDB*, 2(1):970–981, 2009.
- RKH08. S. Rudolph, M. Krötzsch, and P. Hitzler. All elephants are bigger than all mice. In *Description Logics*, 2008.
- SM96. E. Salvat and M.-L. Mugnier. Sound and Complete Forward and Backward Chainings of Graph Rules. In *ICCS'96*, volume 1115 of *LNAI*, pages 248–262. Springer, 1996.
- Sow84. J. F. Sowa. *Conceptual Structures: Information Processing in Mind and Machine*. Addison-Wesley, 1984.