Conjunctive Query Answering Under Existential Rules - Decidability, Complexity, and Algorithms

Michaël Thomazo

Collège Doctoral du Languedoc-Roussillon

THÈSE

Pour obtenir le grade de

Docteur

Délivré par l'**Université Montpellier II**

Préparée au sein de l'école doctorale **I2S**
Et de l'unité de recherche **LIRMM**

Spécialité: **Informatique**

Présentée par **Michaël Thomazo**

Conjunctive Query Answering Under Existential Rules

Decidability, Complexity, and Algorithms

Soutenue le 24 octobre 2013 devant le jury composé de :

Co-directeurs de thèse		
M. Jean-François BAGET	Chargé de recherche	Inria
Mme Marie-Laure MUGNIER	Professeur	Université de Montpellier
Rapporteurs		
M. Georg GOTTLOB	Professeur	Université d'Oxford
M. Carsten LUTZ	Professeur	Université de Brême
Mme Marie-Christine ROUSSET	Professeur	Université de Grenoble
Examinateur		
M. Christophe PAUL	Directeur de recherche	CNRS

UNIVERSITÉ MONTPELLIER 2
SCIENCES ET TECHNIQUES

Contents

List of Figures

List of Tables

Remerciements

Derniers moments avant la soutenance, me voilà à repenser à ces trois années si riches en rencontres et en expériences. Des années agréables, au cours desquelles j'ai eu la chance d'avoir un encadrement humain et scientifique dont on peut à peine rêver. Merci, Jean-François, Marie-Laure d'avoir été des directeurs de thèse parfaits : la porte ouverte lorsqu'il y en a besoin, et quand il le faut, la paix aussi ! Un duo de directeurs complémentaires, avec un seul son de cloche quand on en arrive à décider que faire... le rêve ! Du fond du coeur, merci.

Merci également à Georg Gottlob, Carsten Lutz et Marie-Christine Rousset d'avoir accepté de rapporter ma thèse. Vos nombreuses remarques m'ont permis d'améliorer significativement la qualité de ce manuscrit, et j'espère avoir de nouveau l'occasion de discuter avec vous au cours de prochains conférences, réunions de projets, ou peut-être dans d'autres cadres encore ! Merci également à Christophe Paul d'avoir bien voulu être examinateur, après avoir déjà participé à mon comité de suivi de thèse au cours des trois dernières années.

Pas encadrants, et qui pourtant m'ont beaucoup aidé à me cadrer, je dois aussi remercier les M&M&M's (Madalina, Michel et Michel), pour les "pauses" cafés, les discussions à n'en plus finir, et bien entendu, pour animer avec toujours autant d'entrain les réunions d'équipe. Equipe, toujours, comment oublier Bruno, co-bureau atemporel, qui m'a accompagné au sein du labo comme en dehors, en France comme à l'étranger ! Merci également à Annie, dont l'accueil transforme les formalités administratives en un plaisir (très bonne idée, les Pyrénéens, dans ton bureau !). Bien que je ne t'ai pas facilité la tâche au cours de ces trois années entre mes divers déplacements et mon étourderie certaine, tu as continué à me recevoir avec le sourire : merci ! Merci à toute l'équipe GraphIK, des permanents aux stagiaires, en passant bien sûr par les doctorants, les anciens, les actuels et les futurs, pour être ce qu'elle est : une équipe, à mon sens du terme. Au plaisir de vous revoir, bientôt :)

Le LIRMM ne s'arrête cependant pas à l'équipe, et nombre d'autres personnes m'ont accompagné au sein du LIRMM : merci donc à mes compères du Semindoc, Romain (tes affiches font encore la une du labo ;)) et Fabien, pour avoir formé un trinôme au poil. Merci à Petr et Thibaut – binôme parfait pour se changer les idées au besoin ! Merci à Petit Bousquet pour toujours être d'humeur joyeuse, et à Marthe pour les nombreux fruits, gâteaux et thés... Plus généralement, merci aux AlGCo pour entretenir la bonne humeur du couloir ! Arriver au LIRMM veut aussi dire passer par l'accueil, et à ce poste, comment ne pas remercier Laurie et Nicolas ? Sourire, soutien, efficacité... tout est au rendez-vous ! Merci toujours, à l'administration du LIRMM, avec une pensée spéciale pour Caroline, Cécile et Elisabeth – toujours là pour aider quand le besoin s'en fait sentir. Merci, enfin, à la bande d'ECAI'12 – vous vous reconnaîtrez si vous en avez fait partie ! Beaucoup de souvenirs pour une aventure commune inoubliable.

De-zoomons encore un peu plus, car mon environnement de recherche ne s'arrête pas aux mur du LIRMM: remercier toutes les personnes croisées en conférence pour leur accueil chaleureux serait une entreprise vouée à l'échec – je me contenterai donc d'un merci global à eux, et d'un particulier pour les deux chercheurs qui m'ont hébergé durant deux mois chacun: Ken Kaneiwa, à l'université d'Iwate à l'époque (maintenant à The University of Electro-Communications, Tokyo), et Sebastian Rudolph, au KIT à l'époque, et maintenant à la TU Dresden. Ces deux séjours m'ont beaucoup apporté, que ce soit d'un point de vue scientifique ou humain. Merci à vous ! Merci également aux institutions ayant financé ces séjours : la Japan Society for the Promotion of Science (JSPS) pour mon séjour chez Ken Kaneiwa, à travers son excellent programme d'été (ainsi que le CNRS pour la sélection), et la Deutscher Akamedischer Austausch Dienst (DAAD) pour le séjour chez Sebastian Rudolph. Et un deuxième merci à Sebastian, pour ses nombreuses idées, son enthousiasme inébranlable, et la confiance qu'il me témoigne. Que notre collaboration fleurisse !

Une mention spéciale pour Fabienne, du Petit Grain, qui m'a fourni un "bureau" au calme et d'excellents caféS (la majuscule est volontaire) durant la période de rédaction.

Dernière prise de recul : merci à tous ceux qui m'ont accompagné, de près ou de moins près, au cours de ces trois années passées à Montpellier. Rencontrés au labo, à l'escrime, dans un bar, dans un parc, rencontré car c'est l'ami d'une amie d'un ami, que sais-je encore. Merci donc à toutes ces personnes qui m'ont fait me sentir, finalement, chez moi à Montpellier. Je n'ai pas le coeur à oublier quelqu'un dans cette liste (ni même, à vrai dire, de faire une liste) : que ceux qui doivent y être m'invitent donc à prendre un verre, et je pourrais les remercier de vive voix ! Merci, enfin, à la famille (Papa, Maman, Papy, Mamie, Fab, Debi, Babeth, Jean-Louis, Nico et Benji).

Merci, et à bientôt !

CHAPTER

1 Introduction

Exploiting domain knowledge in an automatic way is currently a challenge that draws a lot of attention from both industry and academy. The interest of *formal ontologies*, which allow to represent this knowledge as logical theories, thus enabling to do reasoning on it, is now widely recognized. In the context of the Semantic Web, systems and applications based on formal ontologies are in particular made possible by the definition of standards of the W3C, such as OWL (Web Ontology Language) and OWL2.

Existing languages for representing formal ontologies such as OWL/OWL2 are heavily based upon *Description Logics* (DLs). Description Logics are a family of languages introduced in the 80's, that can be seen as decidable fragments of first-order logic. In DLs, domain knowledge is represented by concepts, which are unary predicates in the vocabulary of first-order logic, and roles, which correspond to binary predicates. Several constructors, depending on the considered DL, are available to build complex concepts and roles from atomic ones. Relationships between concepts and roles are stated by means of concept (or role) inclusion axioms. Last, individual elements can be defined, and properties (*e.g.*, of which concept they are an instance) of these individuals may be stated. Historically, most of the studies carried out in DLs have focused on ontologies in themselves, like checking their consistency or classifying a concept with respect to a set of predicates. A guiding principle of the design of DLs has been the search of a good trade-off between the expressivity of the considered DL, and the theoretical complexity of the associated reasoning problems.

The recent years have been marked by a great increase of the volume and the complexity of available data. The need for an ontological layer on top of data and for efficient querying mechanisms able to exploit this knowledge has thus become more and more acute. A new reasoning issue, known as *ontology-based data access* (*OBDA*), has been brought up: how to query data while taking ontological knowledge into account? Let us give a small example of what we mean by this expression. Assume that a supermarket

opens an online shop. It may sell for instance tofu, lemonade and wasabi-flavored chocolate. The supermarket wants to help customers to find what they are looking for – and enables a semantic search on its catalog. For instance, a vegetarian customer may be especially interested in products that are good protein sources. A direct search on "protein source" and "vegetarian" will not yield any answer, since these terms do not appear in the product descriptions. Using domain knowledge, one can infer that tofu is suitable for a vegetarian, and moreover contains a high share of proteins. This is the kind of inferences that we want to perform when querying data while taking ontologies into account.

Let us stress the three important components that appear in this example. First, we are given an ontology (tofu is a source of proteins,etc), which describes some domain knowledge. In this dissertation, we made the choice to represent the ontology by means of *existential rules*. Existential rules are positive rules, *i.e.*, they are of the form $B \rightarrow H$, where B and H are conjunctions of positive atoms. No functional symbols appear in existential rules – except for constants – but the head of a rule may contain variables that do not appear in the body, and such variables are existentially quantified. The ability to describe individuals that may not be already present in the knowledge base has been recognized as crucial for modeling ontologies and enable incomplete description of data. Such an ability is not granted within, for instance, Datalog, the language of deductive databases [Abiteboul *et al.*, 1994] but it is in DLs. Existential rules have the same logical form as conceptual graph rules [Sowa, 1984; Chein and Mugnier, 2008], as well as tuple-generating dependencies (TGDs) [Abiteboul *et al.*, 1994], which can be seen as Datalog rules extended with existentially quantified variables in the rule head and are one of the building blocks of the recently introduced Datalog± framework [Calì *et al.*, 2012]. Let us also mention that existential rules generalize lightweight DLs, which are the most studied DLs in the context of OBDA. Data (the supermarket sells tofu,etc) can be represented using different representational models, such as relational databases, graph databases,etc. We will abstract ourselves from such representations and view data as a formula in first-order logic – more precisely, an existentially closed conjunction of atoms. Last, the queries we consider are *conjunctive queries*: find a product that is suitable for a vegetarian and is a source of proteins. These queries are considered as basic in the database community: they are both efficiently processed by relational database management systems and often used. Note that all our results can be extended to unions of conjunctive queries (which can be seen in a logical way as a disjunction of conjunctive queries), since to answer a union of conjunctive queries one can simply evaluate each conjunctive query separately. This problem is a particular instance of the ontology-based data access problem, that we will denote by the name *ontology-based query answering* (OBQA).

In this dissertation, we address the OBQA problem from a theoretical point of view. The presence of existentially quantified variables in the head of rules make this problem undecidable. Classical questions are then the following: what are the most expressive decidable classes of rules that we can design, while keeping decidability of the OBQA problem? Beyond decidability, we are also interested in the worst-case complexity of the problem. Last, we also want to design efficient algorithms for classes of rules that are decidable. The usual approaches to deal with these problems is to use one of the

two following approaches, which we describe here intuitively. A first approach is to "saturate" data, that is, to enrich data in order to add all the information that is entailed by the original knowledge base. A conjunctive query is then evaluated against the saturated data without considering the ontology anymore. This kind of techniques will be called materialization-based in this dissertation. Another popular approach is to reformulate the query into a first-order query using the ontology, and to evaluate this rewritten query directly against the original data. This latter approach is particularly well-suited when the data is so large that materializing all the consequences that can be obtained from the ontology is not reasonable. Moreover, in some cases, the saturated data may be infinite, which prevents from materializing it. However, a similar problem may occur with this materialization-avoiding approach, where no (finite) first-order formula would be a sound and complete rewriting of the original query with respect to the ontology. A more formal description of both approaches, and a landscape of the literature on OBQA are presented in Chapter 2.

The contributions of this dissertation may be split in two parts, according to the two previous kinds of techniques. First, let us give a simple example where the saturated data is not of finite size. We consider a single fact, "Bob is a human", and a simple ontology, stating that every human has a parent that is a human. We do not discuss the empirical soundness of this rule, but let us focus on the facts that are entailed by our knowledge base. Being a human, Bob has a parent that is a human. Let us call this parent x_1. Since x_1 is a human, it also holds that x_1 has a parent that is a human, and we can thus state the existence of an individual x_2, that is both a human and a parent of x_1. This process can be repeated as often as we want, creating infinitely many new individuals. The simplicity of the given ontology advocates that this behavior does not only happen in pathological cases. Since we cannot materialize data of infinite size, are we doomed not to be able to answer queries when such ontologies are considered? The answer is already known, and is no: it is in some cases possible to design algorithms to answer conjunctive queries with respect to data and a set of rules even if the saturated data associated to them is not of finite size. One of these cases is when the structure of the saturated data is close to being a tree. However, for some classes of ontologies that ensure tree-likeness of the saturated data, dedicated algorithms are not known yet. We propose a new criterion, which is a structural condition on the saturated data, that ensures decidability of the conjunctive query answering problem. This class of rules covers a wide range of known decidable classes. The intuition behind this condition is that each rule application adds information on individuals that are "close" one to another. We also provide an algorithm that is worst-case optimal for that class of rules, as well as for most of its known subclasses, up to some small adaptations. To achieve this, we present a way to finitely represent the saturated data. This is done by means of *patterns*. Intuitively, patterns are associated with a set of individuals, and two sets of individuals have equivalent patterns if they are the "root" of similar tree-like structures. This will be developed in Chapter 3.

We also consider materialization-avoiding approaches. A major weakness of most of the rewriting algorithms of the literature is the weak expressivity of the ontologies they support. Indeed, most of them work only for very restricted ontologies. Standing

out is the piece-based rewriting algorithm of [König *et al.*, 2012], that is guaranteed to compute sound and complete rewritings as soon as they exist. However, this algorithm, as well as the majority of others, outputs unions of conjunctive queries. While this is not a restriction in terms of expressivity, we advocate that this is generally not an adequate query language, because of the huge size of the optimal rewritings when large concept and relation hierarchies are present, which is very likely to occur in real-world ontologies. We thus propose to use a more general form of existential positive queries than unions of conjunctive queries, by allowing to use disjunctions in a slightly less restricted way. We also adapt the algorithm of [König *et al.*, 2012] in such a way that it can compute such rewritings, and provide first experiments showing that this approach is more efficient than the classical approach using unions of conjunctive queries. These results are presented in Chapter 4.

We conclude with some possible extensions of this work in Chapter 5.

CHAPTER

2 The landscape of OBQA

Preamble

In this chapter, we present the ontology-based query answering (OBQA) problem, and a landscape of results about it. We first introduce basic vocabulary about facts, queries and ontologies. We explore links between lightweight description logics and existential rules, which are the two mainly used formalisms to express ontologies when studying OBQA. We last present current approaches for answering queries when taking an ontology into account.

As explained in the previous chapter, the problem we consider in this dissertation is called ontology-based query answering (OBQA). We first need a formalization of its three components: facts, queries and ontologies. After explaining how to represent them, and noticing that the OBQA problem is undecidable in the setting we consider, we present the main known decidability criteria and key concepts used to design classes of rules that fulfill these criteria. We then outline the contributions of this dissertation.

2.1 Representing data and queries

A lot of different technologies allow to store data, among which relational databases, graph databases and triple stores. While relational databases are by far the most used until now, other technologies are also relevant and more appropriate in some application domains – depending on the basic operations that need to be performed on the data. In this dissertation, we abstract from a specific language by considering the formalism of *first-order logic*. We use its standard semantics, and assume the reader to be familiar with first-order logic, however we recall below some basic notations that will be used throughout this dissertation.

A logical language $\mathcal{L} = (\mathcal{P}, \mathcal{C})$ is composed of two disjoint sets: a set $\mathcal{P}$ of predicates and a set $\mathcal{C}$ of constants. Please note that we do not consider functional symbols, except for constants. We are also given an infinite set of variables $\mathcal{V}$. Each predicate p is associated with a positive integer, which is called the *arity* of p. A *term* on $\mathcal{L}$ is either a constant (*i.e.*, an element of $\mathcal{C}$) or a variable. As a convention, we will denote constants by $a, b, c, \ldots$ and variables by $x, y, z, \ldots$. An *atom* on $\mathcal{L}$ is of form $p(t_1, \ldots, t_k)$, where p is a predicate of $\mathcal{P}$ of arity k, and t_i is a term on $\mathcal{L}$ for any i. The terms of an atom a are denoted by $\text{terms}(a)$, and its variables are denoted by $\text{vars}(a)$. A *ground atom* contains only constants. The interpretation of a logical language is defined as follows.

Definition 2.1 (Interpretation of a language) *An* interpretation *of a logical language $\mathcal{L} = (\mathcal{P}, \mathcal{C})$ is a pair $\mathcal{I} = (\Delta, .^{\mathcal{I}})$ where Δ is a (possibly infinite) non-empty set called the interpretation domain and $.^{\mathcal{I}}$ is an interpretation function such that:*

1. *for each constant $c \in \mathcal{C}, c^{\mathcal{I}} \in \Delta$;*
2. *for each predicate $p \in \mathcal{P}$ of arity k, $p^{\mathcal{I}} \in \Delta^k$;*

A third condition is sometimes added: two different constants should be interpreted by two different elements of the interpretation domain. This condition is often called the unique name assumption. For the conjunctive query answering problem, it does not change what can be entailed, as long as no equality rules are considered.

We now introduce the notion of fact , which generalizes the usual definition of fact (which is a ground atom), and happens to be convenient in this dissertation.

Definition 2.2 (Fact) *Let $\mathcal{L}$ be a language. A fact on $\mathcal{L}$ is an existentially closed conjunction of atoms on $\mathcal{L}$.*

We extend the notions of terms and variables to facts. In the following, we will usually omit the existential quantifiers in the representation of a fact. However, formulas that represent facts should always be considered as existentially closed. Moreover, we will often consider facts as sets of atoms, hence using inclusion and other set theoretic notions directly on facts. Before introducing other concepts, let us provide some examples illustrating the notion of fact, and clarifying the links between different representations of the same data. This is the purpose of Example 1.

Example 1 *Let F be the following first-order formula:*

$$\exists x \exists y \exists z \; r(x, y) \wedge p(x, z, a) \wedge r(y, z)$$

where a is a constant. F is a fact, since it is an existentially closed conjunctions of atoms.

The queries that we consider are conjunctive queries. Such queries are often written *a la* Datalog:

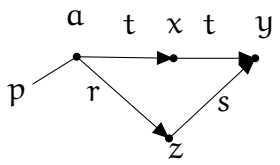

Figure 2.1: Drawing of $p(a) \wedge t(a,x) \wedge t(x,y) \wedge r(a,z) \wedge s(z,y)$

$$q = \text{ans}(x_1, \ldots, x_k) \leftarrow B,$$

where B, called the *body* of q, is a fact, variables $x_1, \ldots, x_k$ occur in B, and ans is a special predicate, which does not appear in any rule, and whose arguments are used to build answers. When $k = 0$, the query is a Boolean query, and it is then only described by its body. In this chapter and in the following, the term query will always denote a Boolean conjunctive query. Restricting ourselves to Boolean queries is not a loss of generality, since conjunctive query answering is classically reducible to Boolean conjunctive query answering.

Facts (and Boolean queries) can be graphically represented, by means of hypergraphs. Terms of F and atoms of F are respectively in bijection with nodes and hyperarcs of the corresponding hypergraph. A node is labeled by the name of its corresponding term, and hyperarcs are labeled by the predicate name of the corresponding atom. When facts are only unary or binary, we will represent them as pictured in Figure 2.1. In particular, (hyper)arcs associated with binary atoms are drawn as arcs from the first to the second argument of the atom.

We now introduce the notion of primal (or Gaifman) graph of a fact, which will be used to lift graph-theoretical notions (such as the treewidth of a graph) to facts. This notion naturally comes from the corresponding notion on hypergraphs: we associate with a fact the primal graph of its hypergraph.

Definition 2.3 (Primal graph) *Let F be a fact. The primal graph $G_F = (V_F, E_F)$ of F, where V_F is the set of vertices of G_F and E_F is its set of edges, is defined as follows:*

- *each term of F is bijectively associated with an element of V_F,*
- *there is an edge between two vertices of V_F if and only if the associated terms appear in the same atom of F.*

Example 2 (Primal graph) *The primal graph of the fact F of Example 1 is given Figure 2.1. It has x, y, z and a as vertices. The only edge which does not exist is the edge between y and a, because no atom of F has both y and a as arguments.*

A basic problem is the entailment problem, which can be expressed on facts as follows: given two facts F_1 and F_2, is it true that F_1 logically entails F_2, *i.e.*, that every model of F_1 is a model of F_2? Logical entailment is denoted by $\models$. It is well known that $F_1 \models F_2$

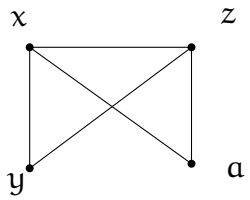

Figure 2.2: The primal graph of $\exists x \exists y \exists z\, r(x,y) \wedge p(x,z,a) \wedge r(y,z)$

if and only if there exists a *homomorphism* from F_2 to F_1. A homomorphism from a fact F to a fact G is a mapping of $\mathrm{vars}(F)$ into $\mathrm{terms}(G)$ such that atoms are preserved.

Definition 2.4 (Substitution, homomorphism, isomorphism) *Let X be a set of variables and T be a set of terms. A* substitution σ *of X by T is a mapping from X to T. Given an atom a, $\sigma(a)$ denotes the atom obtained by substituting each occurrence of $x \in \mathit{vars}(a) \cap X$ by $\sigma(x)$. A homomorphism from a set of atoms S to a set of atoms S' is a substitution of $\mathit{vars}(S)$ by $\mathit{terms}(S')$ such that $\sigma(S) \subseteq S'$.* [1] *In that case, we say that S maps to S' (by σ). An isomorphism from S to S' is a bijective substitution such that $\sigma(S) = S'$.*

A well-known fundamental result is the relation between entailment and homomorphism, which is stated in Theorem 1.

Theorem 1 *Let F and F' be two facts. $F' \models F$ if and only if there exists a homomorphism from F to F'.*

In particular, two facts F and F' are logically equivalent if they are homomorphically equivalent (*i.e.*, there is a homomorphism from one to the other and reciprocally), and this is denoted by $F \equiv F'$. We will sometime refer to the notion of a *core* of a fact.

Definition 2.5 (Core) *A* core *of a fact F, denoted by $\mathit{core}(F)$, is a minimal (with respect to inclusion) subset of F equivalent to F.*

It is well-known that all cores of a fact are isomorphic, and we will thus speak about the core of a fact, which is unique up to isomorphism.

Example 3 *Let $F = r(x,y) \wedge r(x,z) \wedge p(x)$, where quantifiers have been omitted. The core of F is equal (up to isomorphism) to $r(x,y) \wedge p(x)$.*

Another graph-inspired notion that we heavily rely upon is the notion of treewidth. This notion, introduced in [Robertson and Seymour, 1986], can be seen as a measurement of how far a graph is from being a tree. Its definition may be introduced by using the notion of tree decomposition.

1. Recall here that we consider facts as sets of atoms.

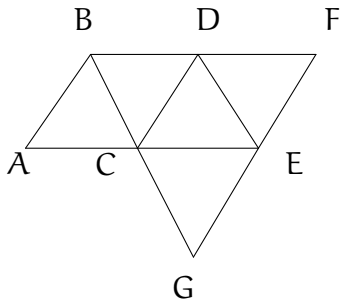

Figure 2.3: A graph

Definition 2.6 (Tree decomposition) *A* tree decomposition *of a graph* $G = (V, E)$ *is a pair* $(\{X_i \mid i \in I\}, T = (I, F))$ *with* $\{X_i \mid i \in I\}$ *a collection of subsets of* V*, called* bags*, and* $T = (I, F)$ *a tree such that:*

1. *for all* $v \in V$*, there exists* $i \in I$ *with* $v \in X_i$*,*
2. *for all* $\{v, w\} \in E$*, there exists* $i \in I$ *such that* $v, w \in X_i$*,*
3. *for all* $v \in V$*, the set* $I_v = \{i \in I \mid v \in X_i\}$ *forms a connected subtree of* T*.*

The third condition is also known as the running intersection property. The width of a tree decomposition $(\{X_i \mid i \in I\}, T)$ *is equal to* $\max_{i \in I} |X_i| - 1$*. The treewidth of a graph* G *is the minimum width of a tree decomposition of* G*.*

The width is defined by $\max_{i \in I} |X_i| - 1$ and not $\max_{i \in I} |X_i|$ in order to ensure that trees have width 1.

Example 4 (Tree decomposition) *Figures 2.3 and 2.4 present a graph and one of its tree decompositions. One can check that any vertex and any edge belong to a bag. The third condition is also fulfilled. For example,* C *belongs to all bags except the (upper)right most, and these bags form a connected graph. The width of the tree decomposition shown in Figure 2.4 is 2.*

In order to illustrate the notion of tree decomposition, we recall a simple property that will be useful in Chapter 3. This property is also a simple way of providing (non-optimal) lower bounds on the treewidth of a graph.

Property 1 *Let* $G = (V, E)$ *be a graph. Let* X *be a clique of* G*, i.e., a subset of* V *such that for any two distinct vertices* x *and* y *of* X*,* $\{x, y\}$ *belongs to* E*. In any tree decomposition of* G*, there exists a bag* B *such that* $X \subseteq B$*.*

Proof: Let $\mathcal{T}$ be a tree decomposition of G. Let $\mathcal{T}_X$ be a minimal subtree of $\mathcal{T}$ such that for any pair $(u, v) \in X \times X$, there is a bag of $\mathcal{T}_X$ that contains both u and v. Let us assume that

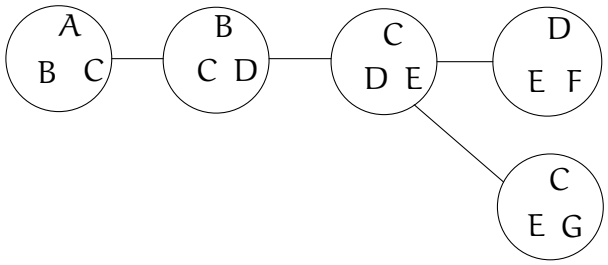

Figure 2.4: A tree decomposition of the graph of Figure 2.3

$\mathcal{T}_X$ is not restricted to a bag. Let B be an arbitrary leaf of $\mathcal{T}_X$. By assumption on $\mathcal{T}_X$, there is x in X such that x is not in B. For any u in $B \cap X$, there exists a bag B_u such that both u and x are in B_u. By the running intersection property, u belongs also to the parent of B. Thus, B could be removed from $\mathcal{T}_X$ while keeping a subtree fulfilling the same condition as $\mathcal{T}_X$, which is a contradiction. Thus $\mathcal{T}_X$ is restricted to a bag, and the claim is proved. □

We finally define the treewidth of a fact.

Definition 2.7 (Treewidth of a fact) *Let F be a fact. The* treewidth *of F, denoted by $tw(F)$, is the treewidth of its primal graph.*

2.2 Ontology formalisms

In this section, we present two formalisms for representing ontologies. First, Description Logics [Baader *et al.*, 2007], the mainstream formalism since the 80's, are introduced. The focus is put on lightweight DLs, which are the most studied DLs in the framework of OBQA. Second, we present existential rules, which are the basic objects considered in this dissertation. Last, we show how lightweight DLs can be expressed by means of existential rules.

2.2.1 Description Logics

While it is out of the scope of this thesis to propose a comprehensive state of the art of the research done on DLs, it is worth to know that, historically, most studies have focused on the analysis of the ontology itself and not on using it for querying data. In this short presentation of DLs, we present $\mathcal{EL}$ and the DL-Lite family, which are called lightweight DLs.

DLs usually encompass two parts: the terminological part, which is called the TBox, and the data, which is called the ABox. The basic objects of a TBox are concepts and roles, which are, in the vocabulary of first-order logic, unary and binary predicates. Properties can be stated about these concepts and roles (such as a role being functional), as

well as relations between roles and concepts. Concepts and roles are inductively defined, starting from atomic concepts and atomic roles. A set of operators allows one to build more elaborated concepts and roles. The expressivity of a DL depends on the set of operators that can be used.

A DL that played a major role in DL research is $\mathcal{ALC}$. Its concepts are built according to the following syntax:

$$C, D \rightarrow A \mid \top \mid \bot \mid \neg C \mid C \sqcap D \mid \forall R.C \mid \exists R.\top,$$

where A is an atomic concept. The formal semantics of $\mathcal{AL}$-concepts is defined by means of *interpretations* $\mathcal{I}$. The logical language of a DL ontology consists in unary and binary predicates (concepts and roles) and constants (in $\mathcal{ALC}$, they appear only in the ABox). An interpretation of such a language is defined as in Definition 2.1, *i.e.* is a non-empty set $\Delta^{\mathcal{I}}$, called the interpretation domain, and an interpretation function $.^{\mathcal{I}}$. That interpretation function assigns to each atomic concept A a subset $A^{\mathcal{I}}$ of $\Delta^{\mathcal{I}}$, and to each atomic role R a subset $R^{\mathcal{I}}$ of $\Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$. The interpretation function is then extended to arbitrary concepts in the following way:

$$\begin{aligned}
\top^{\mathcal{I}} &= \Delta^{\mathcal{I}} \\
\bot^{\mathcal{I}} &= \emptyset \\
(\neg C)^{\mathcal{I}} &= \Delta^{\mathcal{I}} \setminus C^{\mathcal{I}} && \text{(negation of arbitrary concepts)} \\
(C \sqcap D)^{\mathcal{I}} &= C^{\mathcal{I}} \cap D^{\mathcal{I}} && \text{(intersection)} \\
(\forall R.C)^{\mathcal{I}} &= \{a \in \Delta^{\mathcal{I}} \mid \forall b\ (a,b) \in R^{\mathcal{I}} \Rightarrow b \in C^{\mathcal{I}}\} && \text{(universal restriction)} \\
(\exists R.\top)^{\mathcal{I}} &= \{a \in \Delta^{\mathcal{I}} \mid \exists b\ (a,b) \in R^{\mathcal{I}}\} && \text{(existential quantification)}
\end{aligned}$$

More expressive languages could be considered by allowing some other constructors. Those languages are named by adding the corresponding letters to $\mathcal{ALC}$. For instance, one can add:

– union ($\mathcal{U}$):

$$(C \sqcup D)^{\mathcal{I}} = C^{\mathcal{I}} \cup D^{\mathcal{I}}.$$

– full existential quantification ($\mathcal{E}$):

$$(\exists R.C)^{\mathcal{I}} = \{a \in \Delta^{\mathcal{I}} \mid \exists b.(a,b) \in R^{\mathcal{I}} \wedge b \in C^{\mathcal{I}}\}$$

– number restrictions ($\mathcal{N}$):

$$(\geq n R)^{\mathcal{I}} = \{a \in \Delta^{\mathcal{I}} \mid |\{b \mid (a,b) \in R^{\mathcal{I}}\}| \geq n\},$$

and

$$(\leq n R)^{\mathcal{I}} = \{a \in \Delta^{\mathcal{I}} \mid |\{b \mid (a,b) \in R^{\mathcal{I}}\}| \leq n\}.$$

One can also enrich the vocabulary to describe roles. In particular, inverse roles are used, which are denoted by a "-" as superscript, such as in R^-. The influence on semantics is to exchange the first and the second argument of the role, *i.e.*, the interpretation of R^- is $R^{-\mathcal{I}} = \{(a,b) \mid (b,a) \in R^{\mathcal{I}}\}$.

Axioms are then defined by concept and role inclusions, functionality axioms, transitivity axioms, etc. A concept inclusion is of the following form:

$$B \sqsubseteq C,$$

and the associated semantics is:

$$B^{\mathcal{I}} \subseteq C^{\mathcal{I}}$$

Similarly, a role inclusion is denoted by $R \sqsubseteq S$, and the associated semantics is $R^{\mathcal{I}} \subseteq S^{\mathcal{I}}$.

Questions that have been tackled by the DL community focused on this ontological part: is a concept satisfiable, *i.e.*, is it possible for the interpretation of that concept to be non-empty? Is a concept A a subconcept of a concept B, *i.e.*, is it the interpretation of A a subset of the interpretation of B? Are two concepts disjoint?

Some reasoning problems may also be defined when taking the ABox into account, which is a set of assertions on individuals, *i.e.*, a set of statements of the following form: $A(a)$ or $R(a,b)$. Queries concerning the ABox are traditionally restricted to very simple queries that consist of a single ground atom. These queries are called *instance queries*.

Since the problem we consider in this dissertation has been mainly studied for less expressive DLs, we now introduce lightweight description logics. A more detailed and motivated introduction to lightweight description logics can be found in [Baader *et al.*, 2010].

The Description Logic $\mathcal{EL}$

For historical reasons, the first description logics that have been studied all include the possibility to state universal restrictions. However, even with moderately expressive DLs, a basic reasoning problem such as concept inclusion is already PSPACE complete. On the other hand, universal restriction appeared to be far less used than existential restriction in real-world applications.This motivated the study of a novel DL, namely $\mathcal{EL}$ [Baader, 2003]. All roles in $\mathcal{EL}$ are atomic roles. A concept in $\mathcal{EL}$ can be either the top concept (every individual is an instance of the top concept), an atomic concept, the intersection of two concepts, or a concept $\exists r.C$, where r is a role and C an arbitrary concept. A TBox is a finite set of concept inclusions.

An $\mathcal{EL}$ TBox is in normal form if every concept inclusion is of the following form:

$$\begin{aligned} B \sqcap C &\sqsubseteq D, \\ B &\sqsubseteq C, \\ B &\sqsubseteq \exists R.C, \\ \exists R.B &\sqsubseteq C, \end{aligned}$$

where B, C and D are *atomic* concepts. Any $\mathcal{EL}$ TBox can be transformed into a TBox in normal form. This transformation introduces some auxiliary concepts, but entailments on the original vocabulary are preserved [Baader *et al.*, 2005]. This equivalence (up to a given vocabulary) is formalized in Definition 2.29.

$\mathcal{EL}$ has nice computational properties: in particular, the subsumption problem (checking if a concept A is a subconcept of a concept B) is polynomial. On the other hand, it is still expressive enough to cover important practical cases, as can be witnessed by its use to model medical ontologies, such as SNOMED-CT. [2]

The DL-Lite Family

The DL-Lite family has been designed in order to allow efficient conjunctive query answering with large data. The underlying idea is to allow a *reformulation* of the query: TBox axioms are used to reformulate a conjunctive query into a first-order formula that is directly evaluated against a relational database.

We present three members of the family, namely DL-Lite$_{\text{core}}$, DL-Lite$_{\mathcal{F}}$ and DL-Lite$_{\mathcal{R}}$. A basic role Q is either an atomic role P or its inverse P^-. A (general) role R is either a basic role Q or the negation of a basic role $\neg Q$. We present the original DL-Lite [Calvanese *et al.*, 2005]. DL-Lite concepts are defined as follows:

$$\begin{aligned} B &= A \mid \exists R \mid \exists R^-, \\ C &= B \mid \neg B \mid C_1 \sqcap C_2, \end{aligned}$$

where A denotes an atomic concept and R denotes an atomic role; B denotes a basic concept and C denotes a general concept. A DL-Lite$_{\text{core}}$ TBox contains assertions of the following form:

$$B \sqsubseteq C.$$

A DL-Lite$_{\mathcal{F}}$ TBox may also contain functionality axioms, while a DL-Lite$_{\mathcal{A}}$ TBox may contain (basic) role inclusion axioms.

2. http:www.ihtsdo.org/snomed-ct/

2.2.2 Existential Rules

While DLs have been the mainstream formalism to represent and study ontologies from a formal point of view during the last thirty years, other formalisms can be used to represent ontologies. In particular, this is the case of existential rules, which are the main focus of this dissertation. Existential rules – or equivalent objects – have been known under different names: tuple-generating dependencies [Abiteboul *et al.*, 1994], conceptual graph rules [Salvat and Mugnier, 1996], existential rules [Baget *et al.*, 2011a], Datalog$^{\exists}$ rules [Calì *et al.*, 2008],... They form the Datalog± framework [Calì *et al.*, 2009] together with equality rules and negative constraints. The term "Datalog±" makes clear the syntactic similarity between existential rules and Datalog rules: the "+" means that Datalog rules are extended with existentially quantified variables in the head, while the "-" means that rule bodies are syntactically restricted in order to achieve decidability (or low complexity in some cases).

Definition 2.8 (Existential rule) *An* existential rule *(or simply* rule*)* $R = (B, H)$ *on a language* $\mathcal{L}$ *is a closed formula of form* $\forall x_1 \dots \forall x_p (B \rightarrow \exists z_1 \dots \exists z_q \; H)$ *where* B *and* H *are two (finite) conjunctions of atoms on* $\mathcal{L}$*;* $\{x_1, \dots, x_p\} =$ *vars*(B)*; and* $\{z_1, \dots, z_q\} =$ *vars*$(H) \setminus$ *vars*(B)*. Quantifiers are usually omitted, since there is no ambiguity.* B *and* H *are respectively called the* body *and the* head *of* R*, also denoted by* *body*(R) *and* *head*(R)*.*

Rules are used to *infer* new knowledge, starting from an initial fact.

Definition 2.9 (Rule application) *Let* F *be a fact and* $R = (B, H)$ *be a rule.* R *is said applicable to* F *if there is a homomorphism* π *from* B *to* F*. In that case, the* application of R to F according to π *produces a fact* $\alpha(F, R, \pi) = F \cup \pi^{safe}(H)$*, where* $\pi^{safe}(x) = x$ *if* $\pi(x)$ *is defined, and is a "fresh"* [3] *variable otherwise. This rule application is said to be* redundant *if* $\alpha(F, R, \pi) \equiv F$*.*

Example 5 presents an example of rule application.

Example 5 (Rule application) *Let* $F = r(a, b) \wedge p(a) \wedge s(a, c)$ *and* $R = p(x) \rightarrow s(x, y) \wedge q(y)$*.* R *is applicable to* F*, and its application creates the following fact:*

$$r(a, b) \wedge p(a) \wedge s(a, c) \wedge s(a, y_1) \wedge q(y_1),$$

where y_1 *is a fresh existentially quantified variable.*

The notion of frontier of a rule is central in numerous definitions that will be given hereafter. It is the set of variables that are shared by the body and the head of a rule – that is, terms whose image is the set of individuals that are known before the application of the rule and on which (potentially) new knowledge is inferred.

3. A fresh variable is a variable that does not appear yet in any fact or rule.

Definition 2.10 (Frontier of a rule) *Let* $R = (B, H)$ *be a rule. The* frontier *of* R*, denoted by* $fr(R)$*, is the set of variables occurring in both* B *and* H*:* $fr(R) = vars(B) \cap vars(H)$.

Example 6 (Frontier of a rule) *The frontier of* $R = p(x) \wedge s(x,y) \rightarrow s(y,z)$ *is* $\{y\}$.

We mainly consider "pure" existential rules in this dissertation. However, other kinds of rules are of interest, and appear in the Datalog$^{\pm}$ framework: equality rules (also called equality-generating dependencies, EGDs) and negative constraints. We briefly present them here.

Definition 2.11 (Equality rules) *An equality rule* $R = (H, x = t)$ *is a formula of the form* $\forall x_1 \ldots \forall x_p (H \rightarrow x = t)$*, where* x *and* t *are distinct terms, with* $x \in vars(H)$ *and* $t \in vars(H)$ *or is a constant. An application of* R *on a fact* F *is a homomorphism* π *of its body to* F.

We noticed earlier that the unique name assumption does not change the entailments as long as no equality rules are considered. We see here why equality rules may change this: an equality rule may map x and t to two different constants, or t may be a constant and x may be mapped to a different constant. Such an application triggers a *failure*, and the knowledge base is logically inconsistent when a failure is triggered.

A special case of equality rules are functional dependencies, which are widely used in data modeling. Such dependencies may express that for a binary atom, if the value of the first argument is set, then the second is also fixed.

Last, negative constraints express that some fact should not be entailed by the knowledge base. In particular, they allow to express concept disjointness. If a and b are two concepts (unary predicates), one can state that no element can belong to both classes by the following negative constraint:

$$a(x) \wedge b(x) \rightarrow \bot.$$

Is it worth to study existential rules, since DLs are so well established when dealing with ontologies? A good reason to do so it that this shift of formalism allows one to remove some limitations that are inherent to DLs, but whose incidence on the complexity of the problem that we consider should not be taken for granted. This is in particular the case for predicate arity: DLs consider only predicates that are unary or binary, while there is *a priori* no theoretical reason to do so. This is also, and perhaps more importantly, the case when one wants to represent non-tree shaped structures (as would naturally occur in chemistry for example). This yields sufficient reasons to study the OBQA problem not only from a DL point of view, but also from an existential rule point of view.

2.2.3 Translation of lightweight DLs into existential rules

An interesting feature of existential rules is that they allow to translate lightweight description logics, while preserving their semantics. We present here the translation from

DL-Axiom	Translated rule
$B \sqcap C \sqsubseteq D$	$B(x) \wedge C(x) \rightarrow D(x)$
$B \sqsubseteq C$	$B(x) \rightarrow C(x)$
$B \sqsubseteq \exists R.C$	$B(x) \rightarrow R(x,y) \wedge C(y)$
$\exists R.B \sqsubseteq C$	$r(x,y) \wedge B(y) \rightarrow C(x)$

Table 2.1: Translation of (normal) $\mathcal{EL}$-axioms

DL-Axiom	Translated rule
$A \sqsubseteq B$	$A(x) \rightarrow B(x)$
$A \sqsubseteq \exists R$	$A(x) \rightarrow R(x,y)$
$\exists R \sqsubseteq \exists S^-$	$R(x,y) \rightarrow S(z,x)$
$B \sqsubseteq \exists R.C$	$B(x) \rightarrow R(x,y) \wedge C(y)$
$R \sqsubseteq S$	$R(x,y) \rightarrow S(x,y)$
funct(R)	$R(x,y) \wedge R(x,z) \rightarrow y = z$
$B \sqsubseteq \neg C$	$B(x) \wedge C(x) \rightarrow \bot$

Table 2.2: Translation of DL-Lite axioms

these lightweight description logics to existential rules, as can be found in [Calì *et al.*, 2009].

We associate each atomic concept A (resp. basic role R) with a unary (resp. binary) predicate A (resp. R). Translations of axioms from a normal $\mathcal{EL}$ TBox are presented in Table 2.1, while translations of (some) axioms from a DL-Lite Tbox are presented in Table 2.2. The top concept can also be translated by adding rules for each concept (and each role), stating that every instance of that concept is also an instance of the top concept. Please note that only "pure" existential rules are used in the translation of $\mathcal{EL}$ axioms, while equality rules and negative constraints are used to translate functionality and disjointness axioms.

2.3 Tackling an undecidable problem

The core decision problem we consider is the following: given a fact F, a (Boolean) conjunctive query q, and a set of rules $\mathcal{R}$, does it hold that:

$$F, \mathcal{R} \models q?$$

This problem is undecidable [Beeri and Vardi, 1984], even under strong restrictions on sets of rules $\mathcal{R}$ [Baget *et al.*, 2011a]. Indeed, it remains undecidable even with a single rule, or with unary and binary predicates. However, several restrictions in the set of rules are known to ensure decidability. Most of there restrictions can be classified in three

categories. The purpose of this section is to present these categories, as well as brute-force algorithms naturally associated with these categories when possible. We first focus on materialization-based approaches, and then explore materialization-avoiding approaches.

2.3.1 Materialization-based approaches

Definition 2.9 explained how to apply a rule to a fact. This section explains how to use this notion of rule application to solve the OBQA problem. We first give an intuitive idea of approaches that can be developed starting from this notion, and formalize these approaches in a second time. The presented procedure is also known as "chase" in the database community.

Rule applications being defined, a natural way to determine if a query q is entailed by a KB $(F,\mathcal{R})$ is to infer all possible consequences from F and $\mathcal{R}$, by applying rules of $\mathcal{R}$ as long as possible to infer new information. It is then checked whether q can be mapped to the set of atoms thus created. The saturation process is illustrated by Example 7.

Example 7 (Halting forward chaining example) *Let us consider the following two rules:*

- $T : r(x_t,y_t) \wedge r(y_t,z_t) \rightarrow r(x_t,z_t)$*, which states the transitivity of predicate* r*;*
- $R : r(x_r,y_r) \wedge q(x_r) \wedge p(y_r) \rightarrow s(x_r,y_r,z_r)$.

Let F *be the following fact:*

$$F = q(a) \wedge r(a,b) \wedge r(b,c) \wedge r(c,t) \wedge p(t)$$

Applying all possible rules on F *at once results in:*

$$F' = r(a,c) \wedge r(b,t) \wedge F.$$

Indeed, T *can be applied by* π_1 *and* π_2*, where:*

- $\pi_1(x_t) = a, \pi_1(y_t) = b, \pi_1(z_t) = c$,
- $\pi_2(x_t) = b, \pi_2(y_t) = c, \pi_2(z_t) = t$,

creating respectively $r(a,c)$ *and* $r(b,t)$*. On* F*, no more rules are applicable, but we can iterate the same process on* F'*, creating* F''*:*

$$F'' = r(a,t) \wedge F'.$$

Finally, only one rule application (of R*) can bring new information, which yields:*

$$F^* = s(a,t,z) \wedge F''.$$

In the case of Example 7, a finite number of rule applications is sufficient to infer all consequences of the knowledge base. However, this is not always the case, as can be seen with Example 8, where a single rule is responsible for *derivations* (Definition 2.12) of unbounded length.

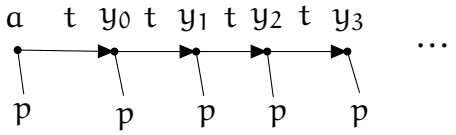

Figure 2.5: An infinite chain

Example 8 *Let* $R' : q(x) \rightarrow s(x,y) \wedge q(y)$. *Let* $F = q(a)$*;* R' *is applicable on* F*, introducing a new atom* $p(y)$*, on which* R' *is again applicable. This yields an infinite chain of atoms, as illustrated in Figure 2.5.*

As we will see later, the non-termination of the saturation process is not a sufficient condition for undecidability of the query answering problem. Specific properties of the so-called *canonical model* may be used in order to ensure decidability even when this canonical model is not finite. In particular, we will be interested in the treewidth of the canonical model. We now formalize the notions that we have presented so far: the notion of derivation, of saturation, as well as so-called canonical model.

Definition 2.12 (Derivation) *Let* F *be a fact and* $\mathcal{R}$ *be a set of 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* i *such that* $1 \leq i \leq k$*, there are a rule* $R = (B, H) \in \mathcal{R}$ *and a homomorphism* π *from* H *to* F_{i-1} *with* $F_i = \alpha(F_{i-1}, R, \pi)$.

When we saturate a fact with respect to a set of rules, it is convenient to apply rules in a breadth-first manner. Indeed, applying rules in an arbitrary order (in a depth-first manner for example) may lead in a loss of completeness of the proposed approach.

Definition 2.13 (k-saturation) *Let* F *be a fact and* $\mathcal{R}$ *be a set or rules.* $\Pi(\mathcal{R}, F)$ *denotes the set of homomorphisms from the body of a rule in* $\mathcal{R}$ *to* F*:*

$$\Pi(\mathcal{R}, F) = \{(R, \pi) \mid R = (B, H) \in \mathcal{R} \text{ and } \pi \text{ is a homomorphism from } B \text{ to } F\}.$$

The direct saturation of F *with* $\mathcal{R}$ *is defined as:*

$$\alpha(F, \mathcal{R}) = F \cup \bigcup_{(R=(B,H),\pi) \in \Pi(\mathcal{R},F)} \pi^{safe}(H)$$

The k*-saturation of* F *with* $\mathcal{R}$ *is denoted by* $\alpha_k(F, \mathcal{R})$ *and is inductively defined by* $\alpha_0(F, \mathcal{R}) = F$ *and* $\alpha_i(F, \mathcal{R}) = \alpha(\alpha_{i-1}(F, \mathcal{R}), \mathcal{R})$ *for* $i > 0$.

Saturating a fact until no new rule application is possible allows us to define the *canonical model* of F and $\mathcal{R}$, also known in the database community as the *universal model*. The particularity of that model is that it maps to any other model of F and $\mathcal{R}$. In particular, to check whether a query q is entailed by F and $\mathcal{R}$, it is sufficient to check if the canonical model of F and $\mathcal{R}$ is a model of q.

Definition 2.14 (Canonical model) *Let F be a fact and $\mathcal{R}$ be a set of rules. The* canonical model *of F and $\mathcal{R}$, denoted by $\alpha_\infty(F,\mathcal{R})$ is defined as:*

$$\alpha_\infty(F,\mathcal{R}) = \bigcup_{k\in\mathbb{N}} \alpha_k(F,\mathcal{R}).$$

Example 9 *By considering Example 7, $\alpha_0(F,\mathcal{R}) = F, \alpha_1(F,\mathcal{R}) = F'$, $\alpha_2(F,\mathcal{R}) = F''$, $\alpha_3(F,\mathcal{R}) = F^* = \alpha_\infty(F,\mathcal{R})$.*

The following theorem is a fundamental tool for solving OBQA.

Theorem 2 ([Baget *et al.*, 2011a]) *Let F be a fact, q a query, and $\mathcal{R}$ be a set of rules. The following properties are equivalent:*

- *$F,\mathcal{R} \models Q$;*
- *there exists a homomorphism from q to $\alpha_\infty(F,\mathcal{R})$;*
- *there is k such that there exists a homomorphism from q to $\alpha_k(F,\mathcal{R})$.*

We now define two abstract properties of sets of rules, related to the structure of canonical models a set of rules generates. Let $\mathcal{R}$ be a set of rules. A first interesting case is when, for any fact F, the canonical model of F and $\mathcal{R}$ is equivalent to a finite fact. In that case, $\mathcal{R}$ is called a finite expansion set.

Definition 2.15 (Finite expansion set) *A set of rules $\mathcal{R}$ is called a* finite expansion set *(*fes*) if and only if, for every fact F, there exists an integer $k = f(F,\mathcal{R})$ such that $\alpha_k(F,\mathcal{R}) \equiv \alpha_\infty(F,\mathcal{R})$.*

The problem of determining if a set of rules is a finite expansion set is undecidable [Baget *et al.*, 2011a] – it is said that finite expansion sets are not *recognizable*. Several known subclasses of *fes* will be presented in the next section. Another abstract class (which is also not recognizable), is the class of bounded treewidth sets. In particular, the canonical model can be of infinite size, as long as it has a tree-like shape.

Definition 2.16 (Bounded treewidth set) *A set of rules $\mathcal{R}$ is called a bounded-treewidth set (*bts*) if for any fact F, there exists an integer $b = f(F,\mathcal{R})$ such that for any $\mathcal{R}$-derivation F' of F, the treewidth of $core(F')$ is less or equal to b.*

Notice that the bound b depends on F, which implies that any finite expansion set is also a bounded treewidth set – it is enough to set b equal to the number of terms of the canonical model of F and $\mathcal{R}$. It has been shown [Calì *et al.*, 2008; Baget *et al.*, 2009],

where the main argument is a result from Courcelle [Courcelle, 1990], that the OBQA problem is decidable when $\mathcal{R}$ is *bts*.

Let us finish this section by presenting a set of rules that is not *bts*.

Example 10 *Let us consider $\mathcal{R}$ containing the following two rules:*
- $R_1 : p(x) \rightarrow r(x,y) \wedge p(y)$,
- $R_2 : r(x,y) \wedge r(y,z) \rightarrow r(x,z)$.

Let F be the following fact: $\{p(a)\}$. R_1 is applicable to F, creating in particular a new atom of predicate p. R_1 can thus be applied infinitely many times. Let us assume that we applied it n times. By applying R_2 until a fix point is reached, the primal graph of the obtained fact is a clique of size $n+1$. Since the canonical model of F and $\mathcal{R}$ contains a clique of size n for any n, its treewidth cannot be bounded.

2.3.2 Materialization avoiding approaches

Another common approach to OBQA relies on a different kind of technique, where no materialization is performed. The main idea is the following: instead of using rules in order to enrich the initial fact, rules are used to rewrite the query. The generated query is then evaluated against the initial fact.

Two kinds of rewritability

One can classify the rewriting techniques existing in the literature in two kinds: rewriting into first-order queries or rewriting into Datalog programs. We successively present both kinds of rewritings.

If Φ is a class of first-order formula, we define the notion of Φ-rewriting, that will be used both in this presentation as well as in Chapter 4.

Definition 2.17 (Φ-rewriting soundness/completeness) *Let Φ be a class of first-order formulas, $\mathcal{R}$ be a set of rules, and q be a conjunctive query. $\varphi \in \Phi$ is a* sound *rewriting of q with respect to $\mathcal{R}$ if for any fact F, it holds that $F \models \varphi$ implies that $F, \mathcal{R} \models q$. φ is a* complete *rewriting of q with respect to $\mathcal{R}$ if for any fact F, the converse holds, that is: $F, \mathcal{R} \models q$ implies that $F \models \varphi$.*

The definition of first-order rewritability corresponds to the existence of such a rewriting for any query – without restricting further the form of the rewriting.

Definition 2.18 (First-order rewritable set of rules) *Let $\mathcal{R}$ be a set of rules. $\mathcal{R}$ is said to be a first-order rewritable set (F.O.R set) if for any query q, there exists a first-order sound and complete rewriting of q with respect to $\mathcal{R}$.*

Another, seemingly more restrictive definition, enforces the rewriting to be a disjunction of conjunctions of atoms, *i.e.*, a union of conjunctive queries (UCQ). We will call such a set of rules a *finite unification set*. Note that this definition is not the original

definition of finite unification sets, which relies on the finiteness of a given rewriting operation. However, we will see in Chapter 4 that the standard definition is equivalent to this one.

Definition 2.19 (Finite unification set) *Let $\mathcal{R}$ be a set of rules. $\mathcal{R}$ is said to be a finite unification set if for any conjunctive query q, there exists a sound and complete UCQ-rewriting (that is, a finite set of conjunctive queries) $\mathcal{Q}$, such that for any fact F, it holds that $F, \mathcal{R} \models q$ if and only if there exists $q' \in \mathcal{Q}$ such that $F \models q'$. Such a set $\mathcal{Q}$ is called a* sound and complete *rewriting of q with respect to $\mathcal{R}$.*

Another strongly related notion has been introduced, called bounded derivation-depth property [Calì *et al.*, 2009]. This notion is presented in the context of a forward-chaining approach, but is highly correlated with finite unification sets.

Definition 2.20 (Bounded derivation-depth property) *Let $\mathcal{R}$ be a set of rules. $\mathcal{R}$ has the* bounded derivation-depth property*, if for any query q, there exists an integer $\gamma = f(q, \mathcal{R})$ such that for any fact F, if $F, \mathcal{R} \models q$, then $\alpha_\gamma(F, \mathcal{R}) \models q$. In particular, γ depends only on q and $\mathcal{R}$.*

This notion may seem quite close to the definition of a finite expansion set (Definition 2.15). However, the bound γ here does not depend on the initial fact F, whereas it could depend on it in the case of *fes*.

As already noted in [Rudolph and Krötzsch, 2013] (for first-order rewritable sets and finite unification sets) these three notions coincide.

Property 2 (Equivalent properties or rule sets) *Let $\mathcal{R}$ be a set of rules. The following three conditions are equivalent:*

1. *$\mathcal{R}$ is a first-order rewritable set,*
2. *$\mathcal{R}$ is a finite unification set,*
3. *$\mathcal{R}$ enjoys the bounded derivation-depth property.*

Not surprisingly, it is undecidable to check if a set of rules is a finite unification set [Baget *et al.*, 2011a].

The other approach uses Datalog programs instead of first-order queries. The literature on Datalog-rewritability has focused on sets of rules that are polynomially rewritable, *i.e.*, that can be rewritten into a program that is of polynomial size in both the query and the set of rules. $\mathcal{EL}$ has been shown to be polynomially Datalog-rewritable [Rosati and Almatelli, 2010; Stefanoni *et al.*, 2012], even though $\mathcal{EL}$-ontologies are not first-order rewritable in general. This is also the case for linear and sticky rules [Gottlob and Schwentick, 2012] (see the next section).

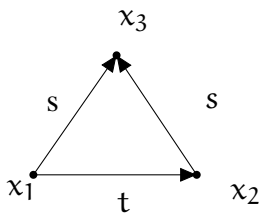

Figure 2.6: The query of the running example

A generic algorithm for first-order rewritable sets

In the remainder of this section, we present a generic algorithm that takes as input any finite unification set $\mathcal{R}$ and any query q, and outputs a sound and complete rewriting of q with respect to $\mathcal{R}$. To illustrate the introduced notions, we will rely on Example 11.

Example 11 *Let* $\mathcal{R}_e = \{R_1, R_2, R_3, R_4, R_5\}$*, defined as follows:*
- $R_1 : p(x) \wedge h(x) \rightarrow s(x,y)$
- $R_2 : f(x) \rightarrow s(x,y)$
- $R_3 : f_1(x) \rightarrow s_1(x,y)$
- $R_4 : t(x,y) \rightarrow t(y,x)$
- $R_5 : s_1(x,y) \rightarrow s(x,y)$

Let q_e *be the following Boolean query:*

$$q_e = t(x_1,x_2) \wedge s(x_1,x_3) \wedge s(x_2,x_3),$$

whose graphical representation is given Figure 2.6.

Example 11 is designed to be both simple to understand and complex enough to illustrate the notions we want to introduce.

The algorithm we present now is based on the notion of *unification* between a query and a rule head. We first recall here the classical definition, as performed by query rewriting approaches (also known as top-down) for (non-existential) Datalog.

Definition 2.21 (Datalog unification) *Let* q *be a conjunctive query, and* R *be a Datalog rule. A unifier of* q *with* R *is a pair* $\mu = (a,u)$*, where* a *is an atom of* q *and* u *is a substitution of* $vars(a) \cup vars(head(R))$ *by* $terms(head(R)) \cup \mathcal{C}$ *s.t.* $u(a) = u(head(R))$.

When a query and a rule unify, it is possible to rewrite the query with respect to that unification, as explained in Definition 2.22.

Definition 2.22 (Datalog rewriting) *Let* q *be a conjunctive query,* R *be a rule and* $\mu = (a,u)$ *be a unifier of* q *with* R*, the rewriting of* q *according to* μ*, denoted by* $\beta(q,R,\mu)$ *is* $u(body(R) \cup \bar{q}')$*, where* $\bar{q}' = q \setminus q'$.

Please note that these classical notions have been formulated in order to stress similarities with the notions we introduce hereafter.

Example 12 (Datalog unification and rewriting) *Let us consider* $q_e = t(x_1,x_2) \wedge s(x_1,x_3) \wedge s(x_2,x_3)$ *and* $R_5 = s_1(x,y) \rightarrow s(x,y)$. *A Datalog unifier of* q_e *with* R_5 *is* $\mu_d = (s(x_1,x_3),\{u(x_1) = x, u(x_3) = y\})$. *The rewriting of* q_e *according to* μ *is the following query:*

$$t(x,x_2) \wedge s_1(x,y) \wedge s(x_2,y).$$

Let us stress that this query is equivalent to the following query:

$$t(x_1,x_2) \wedge s_1(x_1,x_3) \wedge s(x_2,x_3),$$

where x *has been renamed by* x_1 *and* y *by* x_3. *In the following, we will allow ourselves to use such a variable renaming without prior notice.*

Applying the same steps without paying attention to the existential variables in rule heads would lead to erroneous rewritings, as shown by Example 13.

Example 13 (False unification) *Let us consider* $q_e = t(x_1,x_2) \wedge s(x_1,x_3) \wedge s(x_2,x_3)$ *and* $R_2 = f(x) \rightarrow s(x,y)$. *A Datalog unification of* q_e *with* R_2 *is* $\mu_{error} = (s(x_1,x_3), u(x_1) = x, u(x_3) = y)$. *According to Definition 2.22, the rewriting of* q_e *with* R_2 *would be* q_r*:*

$$q_r = t(x,x_2) \wedge f(x) \wedge s(x_2,y).$$

However, q_r *is not a sound rewriting of* q_e*, which can be checked by performing a forward chaining mechanism starting from* q_r *considered as a fact. Indeed,* R_2 *can be applied on* q_r*, creating a new fact* q'_r*:*

$$q'_r = t(x,x_2) \wedge f(x) \wedge s(x_2,y) \wedge s(x,y'),$$

and R_4 *can also be applied, creating* q''_r*:*

$$q''_r = t(x_2,x) \wedge t(x,x_2) \wedge f(x) \wedge s(x_2,y) \wedge s(x,x').$$

No further rule is applicable, and q_e *is not entailed by* q''_r*, which shows that* q_r *is not a sound rewriting of* q_e.

For that reason, the notion of piece unifier has been introduced, originally in the context of conceptual graph rules [Salvat and Mugnier, 1996], then recast in the framework of existential rules [Baget *et al.*, 2011a]. Instead of unifying only one atom at once, one may have to unify a whole "piece", that is, a set of atoms that should have been created by the same rule application. The following definitions and the algorithm are mainly taken from [König *et al.*, 2012].

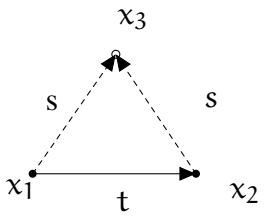

Figure 2.7: x_3 being unified with an existential variable (Example 14), dashed atoms must be part of the unification.

Definition 2.23 (Piece unifier) *Let* q *be a conjunctive query and* R *be a rule. A* piece unifier *of* q *with* R *is a pair* $\mu = (q', u)$ *with* $q' \subseteq q, q' \neq \emptyset$*, and* u *is a substitution of* $fr(R) \cup vars(q')$ *by terms*$(head(R)) \cup \mathcal{C}$ *such that:*

1. *for all* $x \in fr(R)$, $u(x) \in fr(R) \cup \mathcal{C}$ *(for technical convenience, we allow* $u(x) = x$*);*
2. *for all* $x \in sep_q(q'), u(x) \in fr(R) \cup \mathcal{C}$*;*
3. $u(q') \subseteq u(head(R))$*;*

where $sep_q(q')$ *denotes the set of variables that belongs both to* q' *and to* $q \setminus q'$*.*

Let us consider the unification attempted in Example 13 from this point of view.

Example 14 (Piece unifier) *Let us consider* $q_e = t(x_1, x_2) \wedge s(x_1, x_2) \wedge s(x_2, x_3)$ *and* $R_2 = f(x) \rightarrow s(x, y)$. $\mu'_{error} = (q' = \{s(x_1, x_3)\}, u(x_1) = x, u(x_3) = y)$*, defined in Example 13, is not a piece unifier. Indeed,* x_3 *belongs to* $sep_{q_e}(q')$*, and appears in* $s(x_2, x_3)$*, which does not belong to* q'*, violating the second condition of piece unifiers.*

A correct choice of piece is illustrated Figure 2.7. Let $\mu = (\{s(x_1, x_3), s(x_2, x_3)\}, u(x_1) = x, u(x_3) = y, u(x_2) = x)$*.* μ *is a piece unifier of* q_e *with* R_2*, which can be checked by verifying that conditions* 1 *to* 3 *are fulfilled.*

Given that definition of unifiers, the definition of rewritings remains syntactically the same as in the Datalog case.

Definition 2.24 (Rewriting) *Given a conjunctive query* q*, a rule* R *and a piece unifier* $\mu = (q', u)$ *of* q *with* R*, the rewriting of* q *according to* μ*, denoted by* $\beta(q, R, \mu)$ *is* $u(body(R) \cup \bar{q'})$*, where* $\bar{q'} = q \setminus q'$*.*

Example 15 (Rewriting) *Let* μ *be the unifier of* q_e *with* R_2 *defined in Example 14. The rewriting of* q_e *with respect to* μ *is:*

$$\beta(q_e, R_2, \mu) = t(x, x) \wedge f(x).$$

The notion of $\mathcal{R}$-rewriting allows us to denote queries that are obtained thanks to successive rewriting operations.

Definition 2.25 ($\mathcal{R}$-rewriting of q) *Let q be a conjunctive query and $\mathcal{R}$ be a set of rules. An $\mathcal{R}$-rewriting of q is a conjunctive query q_k obtained by a finite sequence $q_0 = q, q_1, \ldots, q_k$ such that for all i such that $0 \leq i < k$, there is $R_i \in \mathcal{R}$ and a piece unifier μ_i of q_i with R_i such that $q_{i+1} = \beta(q_i, R, \mu_i)$.*

We now present the fundamental theorem justifying the notion of $\mathcal{R}$-rewriting. This theorem was originally written in the framework of conceptual graph rules. However, the logical translation of conceptual graph rules is exactly existential rules.

Theorem 3 (Soundness and completeness) *([Salvat and Mugnier, 1996]) Let F be a fact, $\mathcal{R}$ be a set of existential rules, and q be a Boolean CQ. Then $F, \mathcal{R} \models q$ iff there is an $\mathcal{R}$-rewriting q' of q that $F \models q'$.*

Before presenting an algorithm for computing a sound and complete rewriting of a query q with respect to a *fus* $\mathcal{R}$, let us introduce the notion of *original* and *generated* variables in a rewriting.

Definition 2.26 (Original and generated variables) *Let q be a query, $\mathcal{R}$ be a set of rules, and q' be an $\mathcal{R}$-rewriting of q, obtained by a rewriting sequence $q = q_0, q_1, \ldots, q_n = q'$.* Original *variables of q' (with respect to q) are inductively defined as follows:*

- *all variables of q are original;*
- *if q_i has original variables X, and q_{i+1} is the rewriting of q_i with respect to $\mu = (q'_i, u)$, the original variables of q_{i+1} are the images of the elements of X by u.*

A variable that is not original is generated.

We recall that $q_2 \models q_1$ if and only if there is a homomorphism from q_1 to q_2, which we denote by $q_1 \geq q_2$. Let q be a CQ, and $\mathcal{Q}$ be a sound and complete UCQ-rewriting of q. If there exist q_1 and q_2 in $\mathcal{Q}$ such that $q_1 \geq q_2$, then $\mathcal{Q} \setminus \{q_2\}$ is also a sound and complete rewriting of q. This observation motivates the definition of *cover* of a set of first-order queries.

Definition 2.27 (Cover) *Let $\mathcal{Q}$ be a set of conjunctive queries. A* cover *of $\mathcal{Q}$ is a set $\mathcal{Q}^c \subseteq \mathcal{Q}$ such that:*

1. *for any $q \in \mathcal{Q}$, there is $q' \in \mathcal{Q}^c$ such that $q' \geq q$,*
2. *elements of $\mathcal{Q}^c$ are pairwise incomparable with respect to $\geq$.*

Example 16 *Let $\mathcal{Q} = \{q_1 = r(x,y) \wedge t(y,z), q_2 = r(x,y) \wedge t(y,y), q_3 = r(x,y) \wedge t(y,z) \wedge t(u,z)\}$. A cover of $\mathcal{Q}$ is $\{q_1\}$. Indeed, $q_1 \geq q_2$ and $q_1 \geq q_3$, because for $i \in \{2,3\}$, $\pi_{1 \to i}$ is a homomorphism from q_1 to q_i where:*

- $\pi_{1\to 2}(x) = x, \pi_{1\to 2}(y) = \pi_{1\to 2}(z) = y$, *and*
- $\pi_{1\to 3}(x) = x, \pi_{1\to 3}(y) = y, \pi_{1\to 3}(z) = z$.

Algorithm 1 is a generic breadth-first rewriting algorithm, that generates for any query q and any finite unification set $\mathcal{R}$ a sound and complete UCQ-rewriting of q with respect to $\mathcal{R}$. Generated queries are queries that belong to $\mathcal{Q}_t$ at some point; explored queries are queries that belong to $\mathcal{Q}_E$ at some point, and thus, for which all one-step rewritings are generated. At each step, a cover of explored and generated queries is computed. This means that only most general queries are kept, both in the set of explored queries and in the set of queries remaining to be explored. If two queries q_1 and q_2 are homomorphically equivalent, and only q_1 has already been explored, then q_1 is kept and q_2 is discarded. This is done in order not to explore two queries that are comparable by the most general relation – which ensures the termination of Algorithm 1.

Algorithm 1: A BREADTH-FIRST REWRITING ALGORITHM

Data: A *fus* $\mathcal{R}$, a conjunctive query q
Result: A cover of the set of $\mathcal{R}$-rewritings of q
$\mathcal{Q}_F := \{q\}$; *// resulting set*
$\mathcal{Q}_E := \{q\}$; *// queries to be explored*
while $\mathcal{Q}_E \neq \emptyset$ **do**
  $\mathcal{Q}_t := \emptyset$; *// queries generated at this rewriting step*
  for $q_i \in \mathcal{Q}_E$ **do**
    for $R \in \mathcal{R}$ **do**
      for μ *piece-unifier of* q_i *with* R **do**
        $\mathcal{Q}_t := \mathcal{Q}_t \cup \beta(q_i, R, \mu)$;
  $\mathcal{Q}^c :=$ cover$(\mathcal{Q}_F \cup \mathcal{Q}_t)$;
  $\mathcal{Q}_E := \mathcal{Q}^c \setminus \mathcal{Q}_F$; *// select unexplored queries from the cover*
  $\mathcal{Q}_F := \mathcal{Q}^c$;
return $\mathcal{Q}_F$

Let us execute step by step Algorithm 1 on the running example.

Example 17 *Initially,* $\mathcal{Q}_F = \mathcal{Q}_E = \{q_e = t(x_1,x_2) \wedge s(x_1,x_3) \wedge s(x_2,x_3)\}$. *Since* $\mathcal{Q}_E$ *is not empty, we initialize* $\mathcal{Q}_t$ *to the empty set, and consider every element of* $\mathcal{Q}_E$. *The only element of* $\mathcal{Q}_E$ *is* q_e, *so we add to* $\mathcal{Q}_t$ *all possible rewritings of* q_e. *These are:*

- $q_1 = t(x,x) \wedge p(x) \wedge h(x)$, *(by unifying) with respect to* $\mu_1 = (\{s(x_1,x_3), s(x_2,x_3)\}, u_1(x_1) = u_1(x_2) = x, u_1(x_3) = y)$, *unifier of* q_e *with* R_1*;*
- $q_2 = t(x,x) \wedge f(x)$, *with respect to* $\mu_2 = (\{s(x_1,x_3), s(x_2,x_3)\}, u_2(x_1) = u_2(x_2) = x, u_2(x_3) = y)$, *unifier of* q_e *with* R_2*;*
- $q_3 = t(x_2,x_1) \wedge s(x_1,x_3) \wedge s(x_2,x_3)$ *with respect to* $\mu_3 = (\{t(x_1,x_2), u_3(x_1) = y, u_3(x_2) = x)$, *unifier of* q_e *with* R_4*;*
- $q_4 = t(x_1,x_2) \wedge s_1(x_1,x_3) \wedge s(x_2,x_3)$ *with respect to* $\mu_4 = (\{s(x_1,x_3)\}, u_4(x_1) = x, u_4(x_3) = y)$, *unifier of* q_e *with* R_5*;*
- $q_5 = t(x_1,x_2) \wedge s(x_1,x_3) \wedge s_1(x_2,x_3)$ *with respect to* $\mu_5 = (\{s(x_2,x_3)\}, u_5(x_2) = x, u_5(x_3) = y)$, *unifier of* q_e *with* R_5*;*

– $q_6 = t(x,x) \wedge s_1(x,x_3)$, *with respect to* $\mu_6 = (\{s(x_1,x_3), s(x_2,x_3)\}, u_6(x_1) = u_6(x_2) = x, u_6(x_3) = y)$, *unifier of* q_e *with* R_5*;*

Thus $Q_t = \{q_1, q_2, q_3, q_4, q_5, q_6\}$. Q^c *is set to* $\{q_e, q_1, q_2, q_3, q_4, q_5, q_6\}$*, since none of the generated queries are comparable.* Q_E *is set to* $\{q_1, q_2, q_3, q_4, q_5, q_6\}$ *and* Q_F *to* Q^c.

Algorithm 1 performs once more the while loop. Q_t *is reinitialized to the empty set, and all rewritings of* Q_E *are rewritten. We thus explore every* q_i *for* $i \leq 5$. q_1 *and* q_2 *are not unifiable with any rule. We then explore rewritings of* q_3*. The following queries can be obtained by a one step rewriting of* q_3*:*

$$q_1^3 = t(x,x) \wedge p(x) \wedge h(x),$$

$$q_2^3 = t(x,x) \wedge f(x),$$

$$q_3^3 = t(x_1,x_2) \wedge s(x_1,x_3) \wedge s(x_2,x_3),$$

$$q_4^3 = t(x,x) \wedge s_1(x,x_3),$$

$$q_5^3 = t(x_2,x_1) \wedge s_1(x_1,x_3) \wedge s(x_2,x_3),$$

$$q_6^3 = t(x_2,x_1) \wedge s(x_1,x_3) \wedge s_1(x_2,x_3).$$

As for q_4*, the following rewritings are generable:*

$$q_1^4 = t(x_2,x_1) \wedge s_1(x_1,x_3) \wedge s(x_2,x_3),$$
$$q_2^4 = t(x_1,x_2) \wedge s_1(x_1,x_3) \wedge s_1(x_2,x_3).$$

From q_5*:*

$$q_1^5 = t(x_2,x_1) \wedge s(x_1,x_3) \wedge s_1(x_2,x_3),$$
$$q_2^5 = t(x_1,x_2) \wedge s_1(x_1,x_3) \wedge s_1(x_2,x_3).$$

And from q_6*:*

$$q_1^6 = t(x,x) \wedge f_1(x).$$

As illustrated by Figure 2.8, which explicits subsumption relations among queries, a cover of the queries is $\{q_e, q_1, q_2, q_3, q_4, q_5, q_5^3, q_6^3, q_2^4, q_1^6\}$*, which is the new value of* Q^c*. Note that* q_6 *does not belong to* Q^c*, because the newly generated query* q_2^4 *is strictly more general than* q_6*.* Q_E *is set to* $\{q_5^3, q_6^3, q_2^4, q_1^6\}$*,* Q_F *to* Q^c*, and* Q_E *is explored, entering a new iteration of the while loop. Two queries are generated:*

$$q' = t(x_2,x_1) \wedge s_1(x_1,x_3) \wedge s_1(x_2,x_3),$$

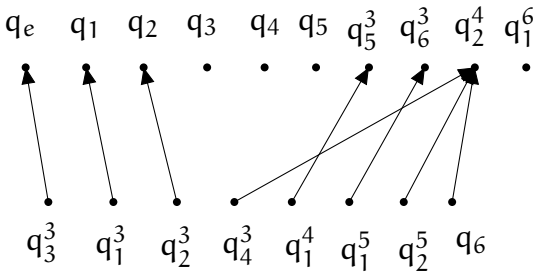

Figure 2.8: There is an arrow from q to q' if and only if q' is more general than q.

and

$$q'' = t(x,x) \wedge f_1(x).$$

At the end of this while loop, we have $\mathcal{Q}_E = \{q'\}$ *and* $\mathcal{Q}_F = \{q_e, q_1, q_2, q_3, q_4, q_5, q_5^3, q_6^3, q_2^4, q_1^6, q'\}$.

Since all queries generable from q' *are covered by* $\mathcal{Q}_F$*, the algorithm halts and outputs:*

$$\{q_e, q_1, q_2, q_3, q_4, q_5, q_5^3, q_6^3, q_2^4, q_1^6, q'\}.$$

The following lemma is crucial for the completeness of Algorithm 1. It ensures that for any queries q and q' such that $q' \geq q$, any rewriting that can be obtained in one step from q is less general than a rewriting that can be obtained in one step from q'. A detailed discussion of what can happen when considering rewriting procedures where this lemma does not hold can be found in [König *et al.*, 2013].

Lemma 1 ([König *et al.*, 2012]) *Let* q_1 *and* q_2 *be two conjunctive queries such that* $q_1 \geq q_2$*. For any rewriting* q_2' *of* q_2 *such that* $q_1 \not\geq q_2'$*, there exists a rewriting* q_1' *of* q_1 *such that* $q_1' \geq q_2'$*.*

Theorem 4 *The output of Algorithm 1 is a sound and complete* $\mathcal{R}$*-rewriting of* q*.*

Proof: We prove by induction on k that for any $i \leq k$, for any q_i that is an i-rewriting of q, there is $q_i^* \in \mathcal{Q}_{F_i}$, that is, Q_F after the i^{th} loop such that $q_i^* \geq q_i$. The only 0-rewriting of q is q, which initially belongs to $\mathcal{Q}_F$, which proves the claim for $k = 0$. Let assume that the claim is true for k, and let us show it for $k+1$. Let q_{i+1} be a $i+1$-rewriting of q. If $i < k$, q_{i+1} is covered by an element of $Q_{F_{i+1}}$ by induction assumption. Otherwise, let q_i be a k-rewriting such that q_{i+1} is a 1-rewriting of q_i. There exists $q_i^* \in Q_{F_i}$ such that $q_i^* \geq q_i$. Lemma 1 ensures that there exists q_{i+1}^* a 1-rewriting of q_i^* such that $q_{i+1}^* \geq q_{i+1}$, which ends the proof. □

2.3.3 Impact of negative constraints and equality rules

We have already mentioned the possibility to add negative constraints and equality rules to the existential rule framework, in order to gain expressivity. What is the effect of such extensions on the decidability of the problem?

As for negative constraints, to add them is inoffensive. If $\mathcal{R}$ is a set of rules for which the OBQA problem is decidable, and $\mathcal{C}_n$ is a set of negative constraints, then the OBQA problem remains decidable when taking both $\mathcal{R}$ and $\mathcal{C}_n$ into account. Indeed, one can check for each constraint, if the body of that constraint is entailed by F and $\mathcal{R}$. If not, then one can safely remove the constraint while keeping logical equivalence. Otherwise, $\bot$ is entailed, and everything is entailed from $F, \mathcal{R}$ and $\mathcal{C}_n$.

The situation is different for equality rules. It has been shown that adding a single equality rule to a *fes, bts* or *fus* may lead to undecidability [Baget *et al.*, 2011a]. To maintain decidability, separability [Calì *et al.*, 2009], whose definition is given Definition 2.28, is to the best of our knowledge the only criterion that has been proposed. The intuition is that equality rules are said separable when they can be considered as constraints that can be checked on the initial database, and then forgotten for the query answering purpose.

Definition 2.28 (Separability [Calì *et al.*, 2009]) *Let $\mathcal{R}$ be a set of existential rules, and $\mathcal{E}$ be a set of equality rules. $\mathcal{E}$ is* separable *from $\mathcal{R}$ iff for any fact F, the following conditions are both satisfied:*

(i) if a failure is triggered during some $(\mathcal{R} \cup \mathcal{E})$-derivation of F, then there is an equality rule applicable to F that triggers a failure;

(ii) if no failure is triggered during any $(\mathcal{R} \cup \mathcal{E})$-derivation of F, then for any BCQ Q, $F, \mathcal{R} \cup \mathcal{E}, \models Q$ if and only if $F, \mathcal{R} \models Q$.

In the remainder of this dissertation, we will not speak further about negative constraints or equality rules. However, negative constraints could freely be added, as well as separable equality rules.

2.3.4 Rule with atomic head: a simplifying assumption

Existential rules may have arbitrarily complex conjunction of atoms as heads. However, it is sometimes useful to assume that they have a very particular form: a single atom. This assumption can be done without loss of generality, as we recall it in this section. More precisely, we show that for any rule set, there exists an "equivalent" rule set for which any rule has an atomic head. We first formalize what equivalent means, since we introduce some auxiliary predicates during the transformation.

Definition 2.29 *Let $\mathcal{P}_1$ be a set of predicates, $\mathcal{R}_1$ and $\mathcal{R}_2$ be two sets of rules. $\mathcal{R}_1$ and $\mathcal{R}_2$ are said* $\mathcal{P}_1$-equivalent *(equivalent when not ambiguous) if for any fact F and query q built on $\mathcal{P}_1$, it holds that $F, \mathcal{R}_1 \models q$ if and only if $F, \mathcal{R}_2 \models q$.*

We now present a transformation [Calì *et al.*, 2008] that takes as input any set $\mathcal{R}$ of rules, built on a set of predicates $\mathcal{P}$, and outputs a set $\mathcal{R}_e$ of rules that are $\mathcal{P}$ equivalent to

$\mathcal{R}$ and have atomic head. Let $R = (B, H)$ be a rule of $\mathcal{R}$. We associate R with $n+1$ rules, *i.e.*, R_b and $\{R^i_h\}_i$, where n is the number of atoms in the head of R, as follows:

- $R_b = B \rightarrow p_R(x)$, where p_R is a fresh predicate (*i.e.*, a predicate that does not belong to $\mathcal{P}$) and x is the set of variables that appear in the head of R;
- $R^i_h = p_R(x) \rightarrow a_i(x)$, where $a_i(x)$ is the i^{th} atom of the head of R.

Property 3 ([Calì *et al.*, 2008]) *$\mathcal{R}$ and $\mathcal{R}_e$ defined as above are $\mathcal{P}$-equivalent.*

2.4 Zoology of concrete decidable classes

We now present "concrete" classes of decidable sets of rules. By "concrete", we mean classes whose recognition problem is decidable. Most of the time, this decision problem is not too complex (in PTIME, or even in NP or co-NP). These classes are based on a restricted number of key concepts, namely: guardedness, backward-shyness and several kinds of acyclicity. Each of these concepts is strongly related with an abstract property we presented in the previous section. Guarded rules (and generalizations of guarded rules) are *bts*, backward-shy rules are *fus*, and different notion of acyclicity lead to sets of rules that are *fes*. Moreover, these concepts have been combined in numerous different ways, yielding a zoology of concrete classes.

Guardedness

Let us start with *guarded* rules. The guarded fragment of first-order logic has been introduced in [Andréka *et al.*, 1998]. It inspired the definition of a guarded rule, which is a rule that possesses a *guard*.

Definition 2.30 (Guard [Calì *et al.*, 2008]) *Let R be a rule. A guard of R is an atom of the body of R that contains all variables of the body. A rule R having a guard is said to be* guarded.

Example 18 (Guard) *Let $R = p(x,y,z) \wedge r(x,y) \wedge q(z) \rightarrow r(y,t)$. R is guarded, since $p(x,y,z)$ is a guard.*
A transitivity rule, such as $r(x,y) \wedge r(y,z) \rightarrow r(x,z)$, is not guarded, since no body atom has x, y and z as arguments.

Most known *bts* classes of rules are based on this notion. The most natural one is the class of guarded rules.

Definition 2.31 (Guarded set [Calì *et al.*, 2008]) *A set of rules $\mathcal{R}$ is guarded if every rule of $\mathcal{R}$ is guarded.*

Guardedness ensures decidability of conjunctive query answering – indeed, a set of guarded rules is *bts*. The original algorithm upper bounds, in function of the set of rules

and the query, the rank of a saturation to which the query is ensured to map if it is entailed by the knowledge base. We will present in the next chapter another algorithm that processes not only guarded sets of rules, but generalizations of them.

A first generalization of guarded rules that one may want to consider is the set of *frontier-guarded* rules, where a guard only needs to contain the frontier of a rule as arguments. These sets are also *bts*. The idea behind this generalization is that only the mappings of terms that belong both to the body and the head of a rule matter, since changing the mapping of other terms would not change the fact obtained after the rule application. These terms correspond exactly to the frontier of the rule.

Definition 2.32 (Frontier-guarded rule [Baget *et al.*, 2010]) *A rule is* frontier-guarded *if there exists an atom of its body that contains every element of the frontier as argument.*

A specific case of frontier-guarded rules is when the size of the frontier is reduced to one element (*fr1*). Such rules generalize in particular Horn DLs that do not contain role inclusion axioms.

Backward shyness

Another key concept is "backward shyness".[4] Backward shyness of a set of rules $\mathcal{R}$ ensures that any query q admits a finite set of $\mathcal{R}$-rewritings. This is a semantic condition that has not been introduced in the literature, but that underlies most of the known *fus* classes of rules.

Definition 2.33 (Backward shyness) *Let $\mathcal{R}$ be a set of rules. $\mathcal{R}$ is to be said backward shy if for any query q, for any $\mathcal{R}$-rewriting q' of q, no generated variable of q' appears in two atoms.*

Property 4 provides an upper bound on the number of most general rewritings of a query q with respect to a backward shy set of rules.

Property 4 (Backward shy rules are *fus*) *Let $\mathcal{R}$ be a set of backward shy rules, and q a query. There are at most $2^{p(|\,terms(q)|+w)^w}$ $\mathcal{R}$-rewritings of q that are not equivalent up to isomorphism, where w is the maximum arity of a predicate and p is the number of predicates appearing in the rules.*

Proof: The number of distinct atoms with arguments the terms of q and at most w other terms is upper bounded by $p(|\,terms(q)|+w)^w$. Since a term that is not a term of q cannot appear in two different atoms, we obtain the claimed upper bound. □

Linear rules [Calì *et al.*, 2008; Baget *et al.*, 2009] are rules whose body contains only one atom. Let us observe that linear rules are backward shy.

4. The name backward shy is from us, and inspired from shy rules [Leone *et al.*, 2012], but there is no inclusion between shy and backward shy rules.

Property 5 *Any set of linear rules is backward shy.*

Proof: The claim follows from the following two remarks:
- when a generated variable is introduced by a rewriting step, it appears in exactly one atom;
- if x appears in k atoms of a query q before a rewriting with respect to $\mu = (q', u)$, then $u(x)$ appears in at most k atoms in the rewriting of q with respect to μ.

□

We now present sticky rules, which have been introduced as a decidability criterion that may deal with non-guarded rules.

Definition 2.34 (Sticky rules [Calì *et al.*, 2010b]) *Let $\mathcal{R}$ be a set of rules. We iteratively mark the variables of the rule bodies of $\mathcal{R}$ according to the following marking procedure. First, for each rule $R \in \mathcal{R}$, and each variable $v \in$ body(R), we mark v if there is an atom a of* head(R) *such that v is not an argument of a. We then apply until a fixpoint is reached the following step: for each rule R, if a marked variable appears in* body(R) *at position (p, i), then we mark for each rule R' each occurrence of the variables of* body(R') *that appear in* head(R') *at position (p, i). $\mathcal{R}$ is said to be* sticky *if there is no rule R such that a marked variable appears more than once in* body(R).

Example 19 provides an example of sticky and non-sticky rules.

Example 19 *Let $\mathcal{R}_1$ be a set of rules containing the following rule:*
- $r(x, y) \wedge t(y, z) \rightarrow s(x, z)$

$\mathcal{R}_1$ is not sticky, since y is marked by the marking procedure, and appears twice in a rule body. On the other hand, the set containing the following two rules is sticky:
- $r(x_1, y_1) \wedge t(y_1, z_1) \rightarrow s(y_1, u_1)$
- $s(x_2, y_2) \rightarrow r(y_2, x_2)$

Indeed, x_1 and z_1 are marked at the initialization step. The propagation step marks y_2, because y_2 appears at the first position of r in the head of the second rule, as x_1 which is already marked. Finally, x_1, z_1 and y_2 are marked, and are the only marked variable. Since none of these variables appears twice in a rule body, this set of rules is sticky.

Property 6 *Any set of sticky rules is backward shy.*

Proof: We show Property 6 by induction on the length of the derivation. If q' is a one-step rewriting of q, then a generated variable is a variable that has been created at this rewriting step. By the initialization step of the sticky marking, such a variable appears at exactly one position, which is marked. Let assume that the induction assumption holds for any k-rewriting of q, and let q' be a $k+1$-rewriting of q. Let q_k be the k-rewriting of q from which q' has been rewritten. A generated variable of q' may appear for two different reasons: either it has been generated at the last rewriting step, and the same reasoning as before can be applied. Or it is a generated variable with respect to q_k. By

induction assumption, it appears at a marked position. The stickiness property implies that it appears also only once in q', and at a marked position. □

Domain-restricted rules [Baget *et al.*, 2011a] are not backward-shy, although they are *fus*. Let us first recall the definition of domain-restricted rules.

Definition 2.35 (Domain-restricted) *A rule* R *is domain-restricted if every atom of the head contains either every variable of the body, or no variable of the body.*

Let us now consider Example 20: it exhibits a set of rules that is domain restricted and not backward shy.

Example 20 *Let* $R : p(x) \wedge q(x) \rightarrow r(x,y) \wedge f(y)$. R *is a domain restricted rule, since* $r(x,y)$ *contains* x *which is the only variable of the body, and* $f(y)$ *does not contain any variable of the body. However, it is not backward shy: by rewriting the query* $f(y)$*, one gets* $p(x) \wedge q(x)$*, where* x *is a generated variable that appears in strictly more than one atom.*

However, it can be understood similarly. Either a rewriting step erases an atom that contains all variables of the body, and no new variable is created, which is a particular case of being backward shy. Or no such atom is erased, and a new connected component, of bounded size, is added. This dichotomy is at the basis of the proof to which we refer the reader [Baget *et al.*, 2011a].

Acyclicity

The last important tool is acyclicity. Several notions based on acyclicity have been defined in the literature: weak-acyclicity, super-weak acyclicity, joint-acyclicity, acyclicity of the graph of rule dependencies, model-summarizing acyclicity, model-faithful acyclicity, weak-recursivity,etc.

Definition 2.36 explicits the construction of the graph of position dependencies.

Definition 2.36 (Graph of position dependencies [Fagin *et al.*, 2005]) *Let* $\mathcal{R}$ *be a set of rules. The (oriented)* graph of position dependencies $(V, A \cup A^*)$ *of* $\mathcal{R}$ *is defined as follows:*

- V *is the set of all positions for all predicates appearing in* $\mathcal{R}$*;*
- *there is an arc from* (p,i) *to* (q,j) *in* A *if there exist a rule* $R \in \mathcal{R}$*, and a variable* $x \in fr(R)$ *such that* x *appears in position* (p,i) *in the body of* R *and in position* (q,j) *in the head of* R*;*
- *there is an arc from* (p,i) *to* (q,j) *in* A^* *if there exist a rule* $R \in \mathcal{R}$ *and a variable* $x \in fr(R)$ *such that* x *appears in position* (p,i) *in the body of* R *and an existentially quantified variable appears in position* (q,j) *in the head of* R*. The arcs of* A^* *are called special arcs.*

The rank *of a position is the maximum number (possibly infinite) of special arcs on a path leading to that position.*

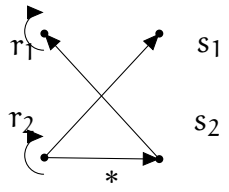

Figure 2.9: The graph of position dependencies associated with $\mathcal{R}$ (Example 21)

Example 21 illustrates the construction of the graph of position dependencies.

Example 21 *Let $\mathcal{R}$ be a set containing the following rules:*
- $r(x,y) \rightarrow s(y,z)$
- $s(x,y) \rightarrow r(y,x)$
- $r(x,y) \wedge r(y,z) \rightarrow r(x,z)$

The graph of position dependencies of $\mathcal{R}$ is shown Figure 2.9. Special arcs are labelled by a star.

This graph is used to define weakly-acyclic sets of rules.

Definition 2.37 (Weak-acyclicity [Fagin *et al.*, 2005]) *Let $\mathcal{R}$ be a set of rules. $\mathcal{R}$ is said to be weakly-acyclic if there exists no cycle in the graph of position dependencies of $\mathcal{R}$ that contains a special arc.*

Let us point out that a set of rules containing no existentially quantified variables in rule heads is trivially weakly acyclic (because there is no special arc). Such a set of rules (which are Datalog programs) are sometimes called *range-restricted*.

Property 7 *A weakly-acyclic set of rules is a finite expansion set.*

The proof is done by upper-bounding for any fact F and any weakly-acyclic set of rules $\mathcal{R}$ the number of fresh existential variables in the core of the canonical model of F and $\mathcal{R}$ (by a double exponential with respect to $\mathcal{R}$; the upper-bound is polynomial if $\mathcal{R}$ is fixed).

Example 22 *In the graph of Figure 2.9 (Example 21), no cycle goes through a special edge, thus $\mathcal{R}$ is weakly acyclic. As such, $\mathcal{R}$ is a finite expansion set. Note that $\mathcal{R}$ does not fulfill any of the criteria viewed so far: the third rule prevents it from being guarded, linear or sticky.*

This condition is sufficient to ensure the finiteness of forward chaining, but not necessary, as witnessed by the following example.

Example 23 *Let* R *be the following rule:*

$$r(x,y) \wedge s(x,y) \rightarrow r(x,v) \wedge r(w,y) \wedge s(x,w) \wedge s(v,y).$$

The graph of position dependencies is a clique of special edges, but an application of R *may not trigger a novel application of* R *– thus,* $\{R\}$ *is a finite expansion set.*

Some generalizations have been proposed in the literature, in particular, super-weak-acyclicity [Marnette, 2009] and join-acyclicity [Krötzsch and Rudolph, 2011]. Both notions are based on the same idea: ensuring that a variable can be propagated from a position occurring in R_1 to a position occurring in R_2 does not ensure that an application of R_1 will trigger an application of R_2. Join-acyclicity refines weak-acyclicity by considering the set of positions in which an existentially quantified variable can be propagated.

Definition 2.38 (Join-acyclicity) *Let* $\mathcal{R}$ *be a set of rules which do not share any variable. For a variable* x*, let* Π_x^B *(resp.* Π_x^H*) be the set of all positions where* x *occurs in the body (resp. head) of a – necessarily unique – rule. For any existentially quantified variable, let* Ω_x *be the smallest set of positions such that:*

1. $\Pi_x^H \subseteq \Omega_x$
2. $\Pi_y^H \subseteq \Omega_x$ *for every universally quantified variable* y *such that* $\Pi_y^B \subseteq \Omega_x$

The existential dependency graph of $\mathcal{R}$ *has the existentially quantified variables of* $\mathcal{R}$ *as its nodes. There is an arc from* x *to* y *if the rule where* y *occurs contains a universally quantified variable* z *with* $\Pi_z^B \subseteq \Omega_x$.

$\mathcal{R}$ *is* jointly acyclic *if its existential dependency graph is acyclic.*

The following example, also from [Krötzsch and Rudolph, 2011], illustrates the difference between weak- and join-acyclicity.

Example 24 *Let* R *be the rule of Example 23:*

$$r(x,y) \wedge s(x,y) \rightarrow r(x,v) \wedge r(w,y) \wedge s(x,w) \wedge s(v,y).$$

$\{R\}$ *is a join-acyclic set of rules, but is not weakly-acyclic. Indeed, the position dependency graph of* $\{R\}$ *is a clique of special edges, and the existential dependency graph of* $\{R\}$ *does not contain any edge.*

The notion of super-weak-acyclicity [Marnette, 2009] is very close in substance to join-acyclicity and generalizes it – though both notions coincide on simple [5] rules [Grau *et al.*, 2012]. We omit the technical definition of super-weak-acyclicity.

Another acyclicity notion, orthogonal to weak-acyclicity, has been proposed based on the notion of rule dependency [Baget *et al.*, 2009] and formerly in [Baget, 2004] for conceptual graph rules. The main idea here is to *characterize* which rule can effectively lead to trigger another rule. Preventing such cycles of dependencies naturally ensures the finiteness of forward chaining.

5. Simple rules are rules where no variable appear twice, and no constant appear.

Figure 2.10: The graph of rule dependencies of Example 25

Definition 2.39 (Dependency) *Let* R_1 *and* R_2 *be two existential rules.* R_2 *depends on* R_1 *if there exist a fact* F*, a homomorphism* π_1 *from* $body(R_1)$ *to* F *and a homomorphism* π_2 *from* $body(R_2)$ *to* $\alpha(F, R_1, \pi_1)$ *such that* π_2 *is not a homomorphism from* $body(R_2)$ *to* F*.*

This definition means that an application of R_1 can trigger a *new* application of R_2. All rule dependencies are summarized in the graph of rule dependencies, whose definition is given below. It is possible to decide if a rule depends on another, by using the notion of a piece-unifier, already introduced.

Definition 2.40 (Graph of rule dependencies) *Let* $\mathcal{R}$ *be a set of rules. The* graph of rule dependencies *of* $\mathcal{R}$*, denoted by* $GRD(\mathcal{R})$ *is defined as follows:*
- *its vertices are the rules of* $\mathcal{R}$*,*
- *there is an arc from* R_1 *to* R_2 *if and only if* R_2 *depends on* R_1*.*

A set of rules $\mathcal{R}$ is said to have an acyclic graph rule of dependencies (aGRD) if $\mathrm{GRD}(\mathcal{R})$ is acyclic. In that case, it is a *fes* as well as a *fus*. This is in particular the case for Example 25. It is worth to note that none of the above mentioned criteria would allow to conclude that this set of rules is a finite expansion set.

Example 25 *Let consider the following two rules:*
- $R_1 = p(x) \rightarrow r(x,y) \wedge r(y,z) \wedge r(z,x)$,
- $R_2 = r(x,y) \wedge r(y,x) \rightarrow p(x)$.

Their graph of rule dependencies is given Figure 2.10.

An (unsuccessful) attempt to generalize both aGRD and weak-acyclicity resulted in some generalizations of aGRD, called aGRD_k. This family of dependencies does not generalize weak-acyclicity, but extends aGRD in an interesting way. In particular, it provides a criterion to determine that the set of rules of Example 26 is a finite expansion set. The interested reader can consult [Baget *et al.*, 2011c].

Example 26 *Let* $R = p(x) \wedge r(x,y) \rightarrow r(y,z)$*.* $\{R\}$ *is a finite expansion set, but is not jointly-acyclic, since the existential dependency graph contains a single node and a loop. Indeed,* $\Omega_z = \{(r,2)\}$*, and the only input position of* y *is* $(r,2)$*, which, according to Definition 2.38 implies the existence of a loop.*

The problem of generalizing both weak-acyclicity and aGRD has recently been solved, by defining *model-faithful acyclicity* and *model-summarizing acyclicity* [Grau *et al.*, 2012]. These semantic notions use a forward-chaining process in order to determine if a set of rules is acyclic or not. The main idea is the following: special predicates mark which rule and which variable have created fresh existentially quantified variables, and sequences of such variable creation are also encoded.

Definition 2.41 (Model-faithful acyclicity [Grau *et al.*, 2012]) *For each rule* $R \in \mathcal{R}$ *where* $R = B[\mathbf{x},\mathbf{y}] \rightarrow H[\mathbf{y},\mathbf{z}]$*, and for each variable* $z_i \in \mathbf{z}$*, let* F^i_r *be a fresh unary predicate unique for* r *and* z_i*; furthermore, let* S *be a fresh binary predicate, and* C *be a fresh nullary predicate. Then* $MFA(R)$ *is the following rule:*

$$B[\mathbf{x},\mathbf{y}] \rightarrow H[\mathbf{y},\mathbf{z}] \wedge \bigwedge_{z_i \in \mathbf{z}} [F^i_r(z_i) \wedge \bigwedge_{y_j \in fr(R)} S(y_j, z_i)].$$

If $\mathcal{R}$ *s a set of rules,* $MFA(\mathcal{R})$ *is the smallest set that contains* $MFA(R)$ *for each rule* $R \in \mathcal{R}$*, as well as the following rule:*

$$S(x_1,x_2) \wedge S(x_2,x_3) \rightarrow S(x_1,x_3),$$

and for every predicate F^i_r*:*

$$F^i_r(x_1) \wedge S(x_1,x_2) \wedge F^i_r(x_2) \rightarrow C.$$

$\mathcal{R}$ *is model-faithful acyclic (MFA) with respect to a fact* F *if* $F \cup MFA(\mathcal{R}) \not\models C$*;* $\mathcal{R}$ *is universally MFA if* $\mathcal{R}$ *is MFA with respect to the all-true fact.*

Checking if a set of rules $\mathcal{R}$ is MFA is a 2ExpTime-complete problem. Thus, a simpler criterion relying on the same idea has also been defined. The only changes relies in the replacement of existentially quantified variables by fresh constants. This notion is called model-summarizing acyclicity (MSA).

MFA generalizes all the notions of acyclicity presented above. A natural question is whether these acyclicity notions capture real-world finite expansion sets. Experiments have been led on a benchmark of 150 ontologies. It happened that 82% of them were MSA; moreover, MSA and MFA were experimentally indistinguishable. The other ontologies of the benchmark are strongly suspected not to be finite expansion sets [Krötzsch, 2012].

Combining tools

Each of the above criteria may be destroyed by adding a single rule (which is not surprising since a single rule can simulate a universal Turing machine). Moreover, if $\mathcal{R}_1$ fulfills a given criterion, and $\mathcal{R}_2$ fulfills another criterion, usually nothing can be said about $\mathcal{R}_1 \cup \mathcal{R}_2$. Table 2.3 sums these incompatibility results up, where "ND" means "not decidable", and "open" that the status of the associated conjunctive query answering problem is not known (to the best of our knowledge).

rr	rr							
linear	ND	linear						
guarded	ND	guarded	guarded					
fr1	ND	fg	fg	fr1				
fg	ND	fg	fg	fg	fg			
dr	ND	ND	ND	open	ND	dr		
sticky	ND	ND	ND	open	ND	ND	ND	
wa	ND	ND	ND	ND	ND	ND	ND	ND
weakly-recursive	ND	ND	ND	ND	ND	ND	ND	ND	ND
	rr	linear	guarded	fr1	fg	dr	sticky	wa	weakly-recursive

Table 2.3: (In-)compatibility results for combinations of decidable classes

However, criteria may be combined in more subtle ways. As an example, the notion of *affected* position, as well as the notion of *glut* variable [Krötzsch and Rudolph, 2011], may be used to restrict the set of variables that need to be guarded.

Definition 2.42 (Affected position) *Let $\mathcal{R}$ be a set of rules. An* affected position *in $\mathcal{R}$ is defined inductively as follows:*

- *if an existentially quantified variable appears at position (p,i), then (p,i) is affected;*
- *if x appears in (p,i) in the head of a rule R, and only in affected positions in the body of R, then (p,i) is affected.*

Glut variables are a refinement of variables that appear at only affected positions, with the intuition that a variable occurring in a rule has to be glut to be mapped to infinitely many existentially quantified variables during a forward chaining procedure. Weakly-guarded (*wg*) rules are then rules that guard all the variables that appear only at affected positions in the body of the rule. One can consider only such variables that belong to the frontier of the rule, which define weakly-frontier-guarded rules (*wfg*). Similar generalization can be defined by using join-acyclicity and glut variables (yielding *jfg* and *glut-fg*). Another specific example of combination are the *weakly-sticky* rules, which are the purpose of Definition 2.43.

Definition 2.43 (Weakly-sticky rules [Calì *et al.*, 2010b]) *Let $\mathcal{R}$ be a set of rules. $\mathcal{R}$ is* weakly-sticky *if and only if for each rule $R \in \mathcal{R}$ and for each variable x that appears more than once in body(R), the following condition holds: x is a non-marked variable, and at least one occurrence of x in body(R) appears at a position of finite rank.*

Please note that weakly-sticky rules do not fulfill any of the the usual criteria that ensure decidability, that is, *fes, fus* and *bts*. Finally, let us mention the recently introduced weakly-recursive rules [Civili and Rosati, 2012a], which are inspired by acyclicity and

backward-shyness notions. It is a *fus* class, which in its current version is incomparable with any other *fus* class.

Both weakly-guarded and weakly-sticky are example of "integrated" combinations: the definitions of the classes themselves are used to create a new class that is also decidable. The graph of rule dependencies (Definition 2.40) allows for more generic combination, assuming that the interactions between different sets of rules are restricted in a proper way. The notion of *directed cut* is the main tool that is used to define "proper interactions".

Definition 2.44 (Directed cut [Baget *et al.*, 2011a]) *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 of $\mathcal{R}_1$ depends on a rule of $\mathcal{R}_2$. It is denoted by $\mathcal{R}_1 \rhd \mathcal{R}_2$ ($\mathcal{R}_1$ "precedes" $\mathcal{R}_2$).*

Property 8 *Let $\mathcal{R}$ be a set of rules such that there exists $\mathcal{R}_1$ and $\mathcal{R}_2$ such that $\mathcal{R}_1 \rhd \mathcal{R}_2$ with $\mathcal{R}_1$* fes *and $\mathcal{R}_2$* bts*. Then $\mathcal{R}$ is* bts*.*

Indeed, one could saturate any fact F by using rules from $\mathcal{R}_1$. This generates a finite fact F'. Rules from $\mathcal{R}_2$ can then be applied, which by assumption generates a fact of bounded treewidth. Since no rule of $\mathcal{R}_1$ depends on a rule of $\mathcal{R}_2$, this ensures that the canonical model of F and $\mathcal{R}_1 \cup \mathcal{R}_2$ is of bounded treewidth. The following property are proven by using the same proof schema.

Property 9 *Let $\mathcal{R}$ be a set of rules such that there exists $\mathcal{R}_1$ and $\mathcal{R}_2$ such that $\mathcal{R}_1 \rhd \mathcal{R}_2$ with $\mathcal{R}_1$ and $\mathcal{R}_2$* fes*. Then $\mathcal{R}$ is* fes*.*

Property 10 *Let $\mathcal{R}$ be a set of rules such that there exist $\mathcal{R}_1$ and $\mathcal{R}_2$ such that $\mathcal{R}_1 \rhd \mathcal{R}_2$ with $\mathcal{R}_1$ and $\mathcal{R}_2$* fus*. Then $\mathcal{R}$ is* fus*.*

Property 11 *Let $\mathcal{R}$ be a set of rules such that there exist $\mathcal{R}_1$ and $\mathcal{R}_2$ such that $\mathcal{R}_1 \rhd \mathcal{R}_2$ with $\mathcal{R}_1$ a* bts *and $\mathcal{R}_2$ a* fus*. Then entailment under $\mathcal{R}$ is decidable.*

These properties, combined with the recognition of some of the concrete classes presented above are at the basis of a tool for analyzing rule sets, called Kiabora. [6] For further details, please consult [Leclère *et al.*, 2013].

Figure 2.11 summarizes known decidable concrete classes. More expressive classes are put higher in the graph. An edge between two classes indicates a (syntactic) subsumption of classes – and the absence of edges indicates that there exists no such subsumption.

6. Available (May 2013) at: http://www2.lirmm.fr/~mugnier/graphik/kiabora/

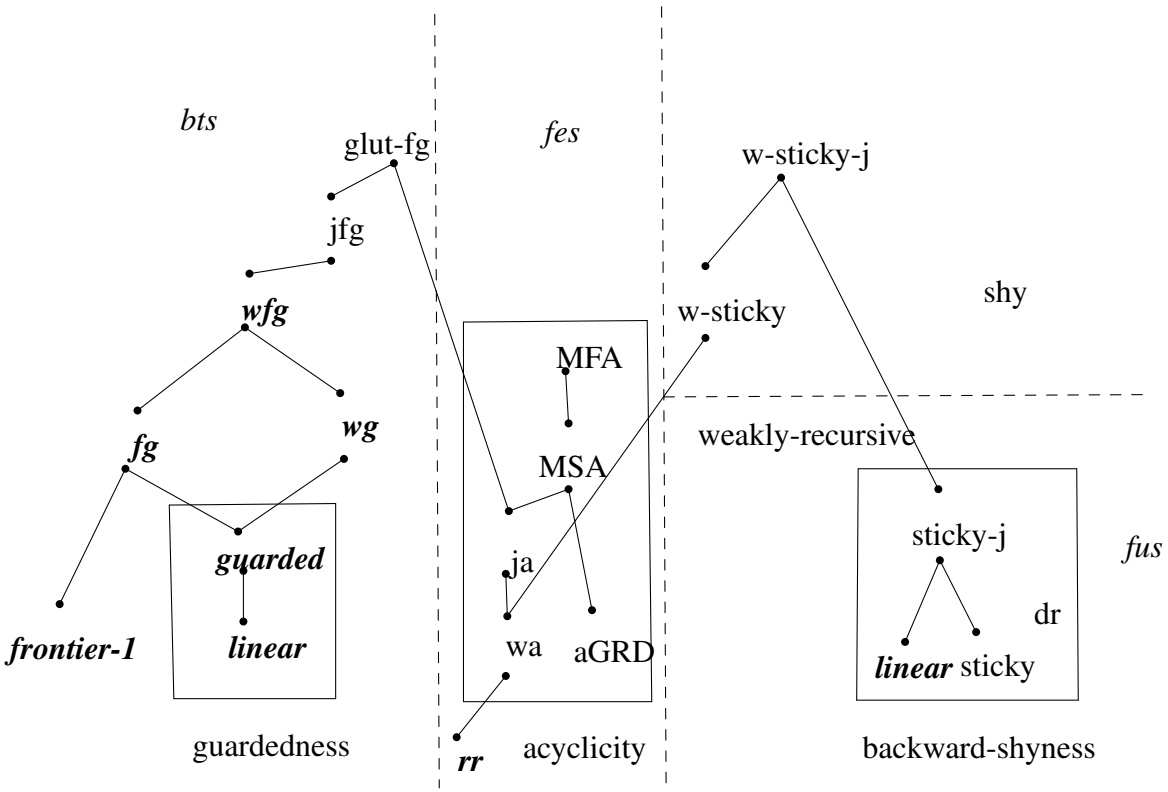

Figure 2.11: A summary of decidable classes.The ***bold italic*** classes belong to *gbts* (see Chapter 3). An edge between a lower and an upper class means that the lower class is a syntactic restriction of the upper class.

Outside of this classification

Some classes that are known to be decidable fall out of the classification we proposed. This in particular the case for *shy* rules (which is the inspiration for the denomination "backward shy"), where existentially quantified variables that are created during a forward chaining process may not be used as joins. The interested reader may consult [Leone *et al.*, 2012].

Another kind of rules is usually problematic: transitivity rules, such as:

$$\mathtt{r}(\mathtt{x},\mathtt{y}) \wedge \mathtt{r}(\mathtt{y},\mathtt{z}) \rightarrow \mathtt{r}(\mathtt{x},\mathtt{z}).$$

Such a rule usually prevents a set of rules containing it to be first-order rewritable. It also hinders the bounded treewidth property, if, for instance, an infinite chain of predicate $\mathtt{r}$ may be built. More generally, complex role inclusions are useful for modeling purposes, but do not fit well in our framework. The problem is actually not only a framework problem: even with lightweight DLs, such as $\mathcal{EL}$, adding arbitrary complex role inclusions makes the conjunctive query answering problem undecidable. However, transitivity (and

more generally *regular* complex role inclusions) can be added to $\mathcal{EL}$ [Krötzsch *et al.*, 2007], by using automata techniques.

We conclude this presentation by presenting a map of decidable classes, as well as a table presenting complexity results for these classes. Combined complexity considers in the input size data, rules and the query. Data complexity considers as fixed both rules and the query. Data complexity is considered in databases to be a better indicator of the complexity of the problem than combined complexity. In particular, a problem whose data complexity is polynomial is often said *tractable*. However, some authors argue that even a data complexity in AC_0 (which is a subclass of LOGSPACE) does not imply practical feasibility for OBQA [Kikot *et al.*, 2011]. Last, we do not present query complexity in the table. Indeed, a lower bound of NP-hardness is easily derivable from a reduction from the three coloration problem, without using a single rule. On the other hand, to the best of our knowledge, no case is yet known where this complexity increase.

Class	**Combined complexity**	**Data Complexity**	Abstract Class
glut-fg	3EXPTIME-c [Krötzsch and Rudolph, 2011]	EXPTIME-h [Krötzsch and Rudolph, 2011]	*bts*
j-fg	2EXPTIME-c [Krötzsch and Rudolph, 2011]	EXPTIME-c [Krötzsch and Rudolph, 2011]	*bts*
wfg	2EXPTIME-c [Baget *et al.*, 2011b]	EXPTIME-c [Baget *et al.*, 2011b]	*bts*
fg	2EXPTIME-c [Baget *et al.*, 2011b]	PTIME-c [Baget *et al.*, 2011b]	*bts*
fr1	2EXPTIME-c [Baget *et al.*, 2011b]	PTIME-c [Baget *et al.*, 2011b]	*bts*
wg	2EXPTIME-c [Calì *et al.*, 2008]	EXPTIME-c [Calì *et al.*, 2008]	*bts*
guarded	2EXPTIME-c [Calì *et al.*, 2008]	PTIME-c [Calì *et al.*, 2008]	*bts*
Datalog (rr)	EXPTIME-c, e.g. [Chandra *et al.*, 1981]	PTIME-c [Dantsin *et al.*, 2001]	*fes,bts*
wa	2EXPTIME-c [Calì *et al.*, 2010a](LB) [Fagin *et al.*, 2005] (UB)	PTIME-c [Dantsin *et al.*, 2001] (LB) [Fagin *et al.*, 2005] (UB)	*fes,bts*
ja	2EXPTIME-c [Krötzsch and Rudolph, 2011]	PTIME-c [Krötzsch and Rudolph, 2011]	*fes,bts*
MFA	2EXPTIME-c [Grau *et al.*, 2012]		*fes,bts*
linear	PSPACE-c [Calì *et al.*, 2010]	in AC_0 [Calì *et al.*, 2009]	*fus,bts*
sticky	EXPTIME-c [Calì *et al.*, 2010a]	in AC_0 [Calì *et al.*, 2010a]	*fus*
weakly-sticky	2EXPTIME-c [Calì *et al.*, 2010a]	PTIME-c [Calì *et al.*, 2010a]	
shy	EXPTIME-c [Leone *et al.*, 2012]	PTIME-c [Leone *et al.*, 2012]	

Table 2.4: Combined and Data Complexities

2.5 Roadmap of contributions

We finish this chapter by giving a high-level roadmap of the contributions of this thesis, as well as some link to other work that have been partly carried out during the Ph.D training.

Chapter 3 presents work done in collaboration with J.-F. Baget, M.-L. Mugnier and S. Rudolph. As already mentioned, some concrete classes that are known to be *bts* do not come with a dedicated algorithm. Our first contribution aims at fixing this lack of algorithm. We introduce a new class of rules, named *greedy bounded treewidth sets*. Where

bts ensures the existence of a tree decomposition of finite width of the canonical model, *gbts* ensures the possibility to actually build one, in a greedy manner. This decomposition could however be of infinite size (i.e., the tree decomposition would contain an infinite number of bags of bounded size). In order to finitely represent this tree, we define an equivalence relation on the bags of the tree decomposition and a copy operation, which takes two equivalent bags of the tree decomposition and copy any child of the first bag under the second bag. Thanks to these tools, we are able to build a finite representation of the canonical model. However, the usual querying operation - homomorphism between graphs - is not complete anymore. We thus introduce an alternative querying operation, that allows us to recover completeness. We show that our algorithm is worst-case optimal for *gbts*, as well as for most of the known subclasses – whose upper-bound on complexity is shown thanks to this algorithm. Related publications are [Baget *et al.*, 2011b; Thomazo *et al.*, 2012; Thomazo, 2012].

Chapter 4 presents work partially done with M. König, M. Leclère and M.-L. Mugnier. It focuses on pure query rewriting approaches, where no materialization is performed. The aim is to design an efficient algorithm for query rewriting under any set of *fus*, and not only specific subclasses. We first consider query rewriting using union of conjunctive queries, and give a characterization of the minimal sound and complete rewriting of a conjunctive query using union of conjunctive queries. After advocating that union of conjunctive queries are not a good tool to express such rewritings when large class or role hierarchies are in the ontology, we present *semi-conjunctive queries*, which are a slightly more general form of positive existential queries. We then propose an algorithm for computing sound and complete rewriting using semi-conjunctive queries, and provide first experiments to evaluate it. Related publications are [König *et al.*, 2012; Thomazo, 2013].

Chapter 5 concludes this thesis, and in particular presents some promising open questions.

Some other questions have been tackled during the last three three years, even though they are not developed in this dissertation. We briefly mention them here. The interested reader is invited to consult related publications. In [Baget *et al.*, 2011c], we consider some weaknesses of the rule dependency notion introduced in [Baget, 2004]. This notion does not behave well with respect to the classical atomic-head decomposition. Indeed, a set of rules may have an acyclic graph of rule dependencies, while the transformed set of rules (where each rule has an atomic head) may have a graph of rule dependencies that is cyclic. This behavior is not desirable, since it weakens the conclusion that can be drawn from the study of the graph of rule dependencies. We thus present a generalization of rule dependency, named *k-dependency*, and show that it keeps some of the decomposition properties of rule dependency, while allowing to overcome that shortcoming. In [Mugnier *et al.*, 2012], we consider conjunctive query answering in the presence of atomic negation, *i.e.*, where both facts and queries may contain negative literals. Even without ontologies, this problem is complete for the second level of the polynomial hierarchy. We introduce

a parameter, the number of so-called *exchangeable literals*, and show that the complexity of the problem drops when this parameter is bounded.

CHAPTER

3 Materialization-based approach

Preamble

In this chapter, we introduce the class of greedy bounded treewidth sets *(*gbts*) of rules. After motivating the definition of that class and explaining links with other known classes, we propose an optimal algorithm for that class, which can be specialized in order to be optimal on the main known subclasses as well.*

The class of bounded treewidth sets (*bts*) is a very expressive class of decidable rules. It covers a wide range of interesting classes, including lightweight description logics and more generally guarded rules. However, as discussed in the previous section, this class of rules have at least two major drawbacks: it is not recognizable, and while numerous concrete classes have been proven to belong to *bts*, and thus have been proven to be decidable, no dedicated algorithm was known prior to the present work. Moreover, the worst-case complexity of these classes of rules was not known. This was the case for frontier-guarded rules as well as for weakly-frontier-guarded rules. Since most of the known *bts* classes were based on the guardedness notion, the goal we set was to extract what seemed to be essential in the guarded property from a graph-theoretic point of view and to design an algorithm solely based on this characteristic. This led to the definition of *greedy bounded treewidth sets* (*gbts*), which is a semantic condition based on the construction of the canonical model by a classical forward-chaining procedure. Greediness enables us to build a finite representation of the canonical model. We create that finite representation by using two tools: a greedy tree decomposition and an equivalence relation on the bags of this tree decomposition. Last, we use a customized querying operation, in order to evaluate queries directly against the finite representation while remaining sound and complete.

A greedy tree decomposition. If a set of rules $\mathcal{R}$ belongs to the *bts* class, then it is ensured that for any fact F, there exists an integer b, such that for any fact F' derived from F and $\mathcal{R}$, there exists a tree decomposition of F' of width at most b. However, this does not provide such a decomposition. Let us consider a set of guarded rules $\mathcal{R}_g$ and try to build such a tree decomposition. We start from a trivial tree decomposition of the primal graph of F, that contains a single bag having as elements every term of F as well as every constant appearing in $\mathcal{R}_g$. If (R_1, π_1) is the first rule application performed, π_1 necessarily maps the frontier terms of R_1 to terms appearing in the initial bag. We can thus add a child to the current tree decomposition (which contains only a root so far): that bag contains every term appearing in the atoms added by the rule application (R_1, π_1), and the resulting structure is a tree decomposition of $\alpha(F, R_1, \pi_1)$. Since $\mathcal{R}_g$ is a guarded set of rules, this process can be repeated as long as we want: the existence of a guard ensures that there exists a bag of the current tree decomposition that contains all terms to which the frontier terms of an applied rule are mapped.

More generally,*i.e.*, for more expressive *gbts* classes, the basic idea is the following: while rules are applied in a breadth-first fashion, a tree decomposition of the current fact is simultaneously maintained. The key point is that this tree decomposition is built in a greedy fashion: new bags of the tree decomposition are created whenever necessary, and choices are never changed. If at some point, it is not possible to extend the tree decomposition in order to take newly created atoms into account, we say that it is not possible to build a greedy tree decomposition of the considered derivation. For *gbts* rules, it is possible to build such a greedy tree decomposition for any derivation.

An equivalence relation. However, building a tree decomposition of the canonical model is not enough to finitely represent this canonical model: indeed, this tree decomposition would also be infinite. However, this additional structure yields natural candidates on which to build an equivalence relation: the bags of the tree decomposition. An important contribution of this dissertation is the design of a correct equivalence relation. We present a first equivalence relation, which is simple and intuitive, that we call *structural equivalence*. However, it happens that structural equivalence does not yield the desired property: two equivalent bags must be the root of "equivalent" subtrees. We thus refine this equivalence notion, using a notion of *pattern of a bag*. With the refined equivalence relation, we can *block* all but one of the bags in an equivalence class. The blocked tree that we obtain is then finite. Moreover, the (possibly infinite) tree decomposition of the canonical model can be built from this blocked tree and the equivalence relation: the blocked tree is said to be *full*.

Creation and evolution rules. The equivalence relation that we propose is however not directly computable: the "natural" way to compute it requires to have already computed the greedy tree decomposition of the canonical model. In order to compute a full blocked tree, we make use of *creation rules* and *evolution rules*. These rules are meant to describe the patterns that may appear in the tree decomposition of the canonical model, and the relationships between patterns. For instance, creation rules intuitively state that any bag

of pattern P that appears in the tree decomposition of the canonical model has a child of pattern P'. We propose such rules, and show how to infer new rules in order to get a complete – but finite – description of the tree decomposition of the canonical model.

A customized querying operation. As presented in the previous chapter, the standard querying operation is homomorphism. However, looking for a homomorphism from a query to atoms of the full blocked tree yields a sound but incomplete algorithm. After exhibiting such a case where completeness is not ensured, we design a new querying operation, by which we can directly evaluate queries on the blocked tree. Intuitively, we guess which structure (this is called an atom-term partition tree) is induced by a homomorphism from the query to the canonical model, and check that it actually corresponds to a homomorphism.

Running example. In order to illustrate the numerous definitions of this section, we will rely on a running example. This example has been designed with the following requirements in mind. First, it should be easy enough to understand. Second, it should illustrate every aspect of our approach, and explain why simpler approaches we could think of are not sufficient. Last, it should not be directly expressible by means of description logics.

Let us introduce the rules of the running example. We first introduce the following three rules:

- $R_1 : q_1(x_1, y_1, z_1) \rightarrow s(y_1, t_1) \wedge r(z_1, t_1) \wedge q_2(t_1, u_1, v_1)$;
- $R_2 : q_2(x_2, y_2, z_2) \rightarrow s(y_2, t_2) \wedge r(z_2, t_2) \wedge q_3(t_2, u_2, v_2)$;
- $R_3 : q_3(t_3, u_3, v_3) \rightarrow h(t_3)$.

These rules are a finite expansion set (since, for example, their graph of rule dependency is acyclic). Applying these rules will create some existentially quantified variables. A first interesting phenomenon is that these existential variables can allow to infer some new information about the initial terms, when combined with the following rules:

- $R_4 : q_2(x_4, y_4, z_4) \wedge s(y_4, t_4) \wedge r(z_4, t_4) \wedge h(t_4) \rightarrow h(x_4) \wedge p_1(y_4) \wedge p_2(z_4)$;
- $R_5 : q_1(x_5, y_5, z_5) \wedge s(y_5, t_5) \wedge r(z_5, t_5) \wedge h(t_5) \rightarrow p_1(y_5) \wedge p_2(z_5)$.

While it can be argued that these rules are slightly complicated, it will allow to illustrate why we cannot block nodes without being careful. Last, we introduce a set of two rules that will generate infinitely many fresh existential variables, and allow us to illustrate both the blocking procedure and the querying operation.

- $R_{p_1} : p_1(x_p) \wedge i(x_p) \rightarrow r(x_p, y_p) \wedge p_2(y_p) \wedge i(y_p)$;
- $R_{p_2} : p_2(x_q) \wedge i(x_q) \rightarrow s(x_q, y_q) \wedge p_1(y_q) \wedge i(y_q)$.

In the remaining of this chapter, $\mathcal{R}_r$ ("r" stands for running) denotes the set containing R_i for $i \leq 5$ as well as R_{p_1} and R_{p_2}. We will consider derivations of the following fact:

$$F_r = q_1(a, b, c) \wedge q_1(d, c, e) \wedge q_1(f, g, g) \wedge i(c) \wedge i(g).$$

Let us take a look at k-saturations of F_r with respect to $\mathcal{R}_r$. On F_r, only R_1 is applicable by three different homomorphisms, creating three sets of three new atoms: $\{s(b, t_1^1), r(c, t_1^1), q_2(t_1^1, u_1^1, v_1^1)\}$,$\{s(c, t_1^2), r(e, t_1^2), q_2(t_1^2, u_1^2, v_1^2)\}$ and

$\{s(g,t_1^3), r(g,t_1^3), q_2(t_1^3,u_1^3,v_1^3)\}$. $\alpha_1(F,\mathcal{R}_r)$ is equal to the union of F_r and these three sets of atoms. On $\alpha_1(F,\mathcal{R}_r)$, three new rule applications are possible, each one mapping the body of R_2 to one of the atoms of predicate q_2. Again, three new sets of atoms are introduced, which are $\{s(u_1^1,t_2^1), r(v_1^1,t_2^1), q_3(t_2^1,u_2^1,v_2^1)\}$,$\{s(u_1^2,t_2^2), r(v_1^2,t_2^2), q_3(t_2^2,u_2^2,v_2^2)\}$ and $\{s(u_1^3,t_2^3), r(v_1^3,t_2^3), q_3(t_2^3,u_2^3,v_2^3)\}$. This yields $\alpha_2(F,\mathcal{R}_r)$. On this fact, three new rule applications of R_3 are possible, which introduce $h(t_2^1), h(t_2^2), h(t_2^3)$. The introduction of these atoms triggers new applications of R_4, creating $h(t_1^1), h(t_1^2), h(t_1^3), p_1(u_1^1), p_1(u_1^2), p_1(u_1^3), p_2(v_1^1), p_2(v_1^2), p_2(v_1^3)$. R_5 is now triggered, creating $p_1(b), p_2(c), p_1(c), p_2(e), p_1(f), p_2(f)$. The union of all atoms considered so far is equal to $\alpha_5(F,\mathcal{R}_r)$, that we will also denote by F_r'. R_{p_1} and R_{p_2} are now applicable, both mapping their frontier to c and g. They will create infinite branches.

3.1 Greedy-bounded treewidth sets

We introduce here the class of rules that will be the focus of this chapter. This class, called *greedy bounded treewidth sets* (*gbts*), contains sets of rules for which all derivations fulfill a given criterion: for every fact F and every $\mathcal{R}$-derivation on F, one should be able to build a tree decomposition by following a fixed algorithm, that we describe hereafter.

We introduce with Definition 3.1 the notion of greedy derivation, that is at the core of the definition of *gbts* rules. Each rule application creates a set of atoms. Intuitively, a derivation is greedy if at each step, the frontier of the applied rule is mapped to terms that are "close" one to another. Terms from the initial fact (and constants appearing in a rule head) are close to any other term. Terms appearing in atoms coming from a single rule head are close one to another.

Definition 3.1 (Greedy Derivation) *An $\mathcal{R}$-derivation $(F_0 = F),\ldots,F_k$ is said to be* greedy *if, for all i with $0 < i < k$, there is $j \leq i$ such that $\pi_i(fr(R_i)) \subseteq vars(A_j) \cup vars(F_0) \cup \mathcal{C}$,* [1] *where $A_j = \pi_j^{safe}(head(R_j))$.*

We present in Example 27 a non-greedy derivation. Figure 3.1 illustrates this example.

Example 27 (Non-greedy derivation) *Let $\mathcal{R} = \{R_0, R_1\}$ where:*
$R_0 = r_1(x,y) \rightarrow r_2(y,z)$ *and*
$R_1 = r_1(x,y), r_2(x,z), r_2(y,t) \rightarrow r_2(z,t)$.
Let $F_0 = \{r_1(a,b) \wedge r_1(b,c)\}$ and $S = F_0,\ldots,F_3$ with:
$F_1 = \alpha(F_0, R_0, \{(y,b)\}), A_0 = \{r_2(b,x_1)\}$,
$F_2 = \alpha(F_1, R_0, \{(y,c)\})$, $A_1 = \{r_2(c,x_2)\}$,
$F_3 = \alpha(F_2, R_1, \pi_2)$, *with* $\pi_2 = \{(z,x_1),(t,x_2)\}$;
$fr(R_1) = \{z,t\}$ is mapped to existential variables by π_2, however there is no A_j s.t. $\{\pi_2(z), \pi_2(t)\} \subseteq vars(A_j)$, thus S is not greedy.

1. We recall that $\mathcal{C}$ denotes the set of constants.

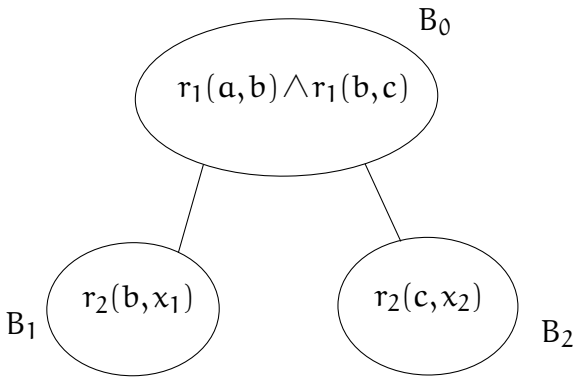

Figure 3.1: Attempt of building a greedy tree decomposition

To any greedy derivation is assigned a so-called derivation tree, formally defined below. Intuitively, the root of the tree corresponds to the initial fact, and each other node corresponds to a rule application of the derivation. Each node is labelled by a set of terms and a set of atoms; note however that the set of atoms is only given to ease understanding, only the set of terms is used in the following.

Definition 3.2 (Derivation Tree - Prefix) *Let* $S = (F_0 = F), \ldots, F_k$ *be a greedy derivation. The* derivation tree *assigned to* S*, notation* $DT(S)$*, is a tree* $T = (\mathcal{X} = \{X_0, \ldots, X_k, \ldots\}, U)$ *such that:*

1. *Let* $T_0 = vars(F) \cup \mathcal{C}$*. The root of the tree is* X_0 *with* $terms(X_0) = T_0$ *and* $atoms(X_0) = atoms(F)$*.*
2. *For* $0 < i \leq k$*, let* R_{i-1} *be the rule applied according to homomorphism* π_{i-1} *to produce* F_i*; then* $terms(X_i) = vars(A_{i-1}) \cup T_0$ *and* $atoms(X_i) = atoms(A_{i-1})$*. There is an edge between* X_i *and the node* X_j *such that* j *is the smallest integer for which* $\pi_{i-1}(fr(R_{i-1})) \subseteq terms(X_j)$*.*

The nodes of $DT(S)$ *are also called* bags*. A* prefix *(subtree) of a derivation tree is a tree is obtained by removing all the descendants of a set of bags.*

Note that, in the second point of the definition, there is at least one j with $\pi_{i-1}(fr(R_{i-1})) \subseteq terms(X_j)$ because S is greedy. Moreover, we choose the smallest one, which means that we link the new bag "as high as possible".

Example 28 *Let us consider the following sequence of rule applications:*

- R_1 *is applied to* F_r *by* π_1 *such that:* $\pi_1(x_1) = a, \pi_1(y_1) = b, \pi_1(z_1) = c$*, creating* $\{s(b, t_1^1), r(c, t_1^1), q_2(t_1^1, u_1^1, v_1^1)\}$*.*
- R_2 *is applied to* $\alpha(F_r, R_1, \pi_1)$ *by* π_2 *such that* $\pi_2(x_2) = t_1^1, \pi_2(y_2) = u_1^1, \pi_2(z_2) = v_1^1$*, creating* $\{s(u_1^1, t_2^1), r(v_1^1, t_2^1), q_3(t_2^1, u_2^1, v_2^1)\}$

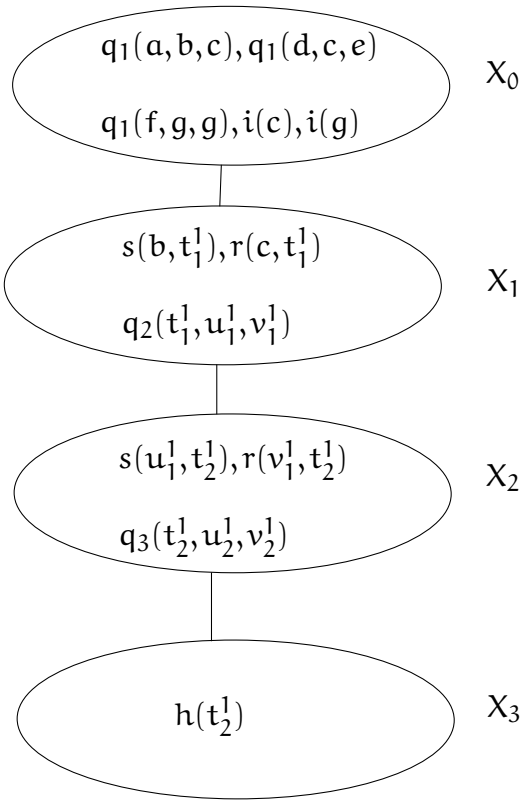

Figure 3.2: The derivation tree associated with Example 28

– R_3 *is applied on the resulting fact by* π_3 *such that* $\pi_3(x_3) = t_1^2, \pi_3(y_3) = u_1^2, \pi_3(z_3) = v_1^2$*, create a single new atom,* $h(t_2^1)$.

This derivation is greedy, and its derivation tree is represented in Figure 3.2.

The following property is easily checked, noticing that T_0 occurs in each bag, which ensures that the running intersection property is satisfied.

Property 12 *Let* $S = F_0 \ldots, F_k$ *be a greedy derivation. Then* $DT(S)$ *is a tree decomposition of* F_k *of width bounded by* $|vars(F)| + |\mathcal{C}| + max(|vars(head(R))|_{R \in \mathcal{R}})$.

Definition 3.3 (Greedy bounded-treewidth set of rules (*gbts*)) $\mathcal{R}$ *is said to be a* greedy bounded-treewidth set *(gbts) if (for any fact* F*) any* $\mathcal{R}$*-derivation (of* F*) is greedy.*

The tree that is greedily associated to a derivation does not depend on the order in which rules are applied. If we assume that no rule application is executed twice, we can thus associate with any fact F and any *gbts* set $\mathcal{R}$ a unique (possibly infinite) derivation tree, that is a tree decomposition of the canonical model of F and $\mathcal{R}$, and that we call the *ultimate derivation tree*.

Property 13 (Ultimate derivation tree) *Let* F *be a fact, and* $\mathcal{R}$ *be a* gbts *set of rules. There exists a unique (up to variable renaming) greedy derivation tree associated with* $\alpha_\infty(F,\mathcal{R})$. *This derivation tree is called the* ultimate derivation tree *of* F *and* $\mathcal{R}$.

The class *gbts* is a strict subclass of *bts* and does not contain *fes* (e.g., in Example 27: $\mathcal{R}$ is *fes* but not *gbts*). It is nevertheless an expressive subclass of *bts* since it contains weakly-frontier-guarded rules (*wfg*):

Property 14 *Weakly-frontier-guarded rules are* gbts.

Proof: Let $\mathcal{R}$ be a *wfg* rule set. Given any $\mathcal{R}$-derivation, consider the application of a rule R_i, with weak frontier-guard g. We want to show that there exists a bag containing the image of any element of the frontier. A variable of the frontier that is not an argument of g is not affected, and thus maps to an initial term, which implies that its image belong to any bag. Let x be an element of the frontier that is also an argument of g. Let $a = \pi_i(g)$. Either $a \in F$ or $a \in A_j$ for some $j \leq i$. In the first case, $\pi_i(x)$ is an initial term, and thus belong to any bag. In the second case, $\pi_i(x)$ is a term if A_j. Thus, the whole frontier maps to the terms of a single bag. We conclude that $\mathcal{R}$ is *gbts*. □

Example 29 $\mathcal{R}_r$ *from the running example is* gbts*: indeed, every rule is frontier-guarded (but* R_4 *and* R_5 *are not guarded).*

The following example shows that *gbts* strictly contains *wfg*.

Example 30 $R = r_1(x,y), r_2(y,z) \rightarrow r(x,x'), r(y,y'), r(z,z'), r_1(x',y'), r_2(y',z')$*;* $\{R\}$ *is* gbts*, but not* wfg *(nor* fes*). First, let us notice that all positions of* r_1 *and* r_2 *are affected, and that* x, y *and* z *belong to the frontier of* R. *Thus,* $\{R\}$ *is not* wfg. *Moreover, let us consider* $F = r_1(a,b) \wedge r_1(b,c)$. R *is applicable to* F*, creating* $r(a,x_1), r(b,y_1), r(c,z_1), r_1(x_1,y_1), r_2(y_1,z_1)$*, as shown in Figure 3.1.* R *is thus newly applicable, mapping its frontier to* x_1 *and* z_1*. This can be repeated infinitely many times, thus showing that* $\{R\}$ *is not* fes*. Last, the only way to map the body of* R *to terms that do not belong to an arbitrary initial fact is to map the frontier to terms that have been created in the same bag, thus ensuring that* $\{R\}$ *is* gbts.

However, *gbts* and *wfg* rules are very close: conjunctive query entailment under *gbts* rules can be polynomially reduced to the same problem under *wfg* rules. We define a mapping τ that translates any knowledge base $\mathcal{K} = (F,\mathcal{R})$ into a knowledge base $\tau(\mathcal{K}) = (\tau(F),\tau(\mathcal{R}))$ such that $\tau(\mathcal{R})$ is *wfg*.

Let F be a fact and $\mathcal{R}$ be an arbitrary set of rules. τ associates F and $\mathcal{R}$ with $\tau(F)$ and $\tau(\mathcal{R})$ defined as follows. We consider two new predicates: a unary predicate f and a q-ary predicate same (with $q = \max(|\text{terms}(\text{head}(R))|_{R \in \mathcal{R}})$. Intuitively, f will mark terms from the initial fact F, as well as constants added by rule applications, and same will gather terms that are "in the same bag". We define then:

- $\tau(F) = F \cup \{f(t) \mid t \in \text{terms}(F)\}$ item $\tau(\mathcal{R}) = \mathcal{R}'_1 \cup \mathcal{R}'_2$, where $\mathcal{R}'_1$ and $\mathcal{R}'_2$ are defined as follows:

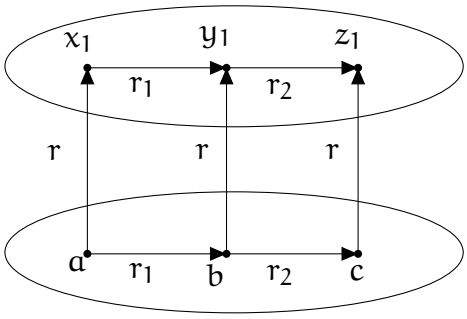

Figure 3.3: Illustration of Example 30

- $\mathcal{R}'_1$ contains rules processing the *same* predicate:
  1. $R'^{1}_1 = f(x) \rightarrow \text{same}(x,\ldots,x)$
  2. $R'^{2}_1 = \text{same}(x_1,x_2,\ldots,x_q) \wedge f(x) \rightarrow \text{same}(x,x_2\ldots,x_q)$
  3. One rule of the following type for each $1 \leq i \leq q$:
     $R'^{3i}_1 = \text{same}(x_1,\ldots,x_i,\ldots,x_q) \rightarrow \text{same}(x_i,\ldots,x_1,\ldots x_q)$
  4. $R'^{4}_1 = \text{same}(x_1,\ldots,x_{q-1},x_q) \rightarrow \text{same}(x_1,\ldots,x_{q-1},x_1)$
- $\mathcal{R}'_2$ contains rules translating rules from $\mathcal{R}$: let $R = B[\boldsymbol{x},\boldsymbol{y}] \rightarrow H[\boldsymbol{y},\boldsymbol{z}]$, and let $c_1,\ldots,c_k$ be the constants in $(H \setminus B)$:
  $\tau(R) = B[\boldsymbol{x},\boldsymbol{y}] \wedge \text{same}(\boldsymbol{y},\boldsymbol{v}) \rightarrow H[\boldsymbol{y},\boldsymbol{z}] \wedge \text{same}(\boldsymbol{y},\boldsymbol{z},\boldsymbol{w}) \wedge_{i:1\ldots,k} f(c_i)$,
  where $\boldsymbol{v}$ and $\boldsymbol{w}$ are sets of fresh variables.

Intuitively, Rules R'^{1}_1 and R'^{2}_1 express that the initial terms (as well as constants added by rule applications) are in all bags; Rules R'^{3i}_1 and Rule R'^{4}_1 respectively allow any permutation and any duplication of arguments in an atom with predicate same . In the translation of the rules from $\mathcal{R}$, the sets of variables $\boldsymbol{v}$ and $\boldsymbol{w}$ are used to fill the atoms with predicate same to obtain arity q.

$\mathcal{R}'_1 \setminus \{R'^{2}_1\}$ is *guarded*. $\mathcal{R}'_2$ is *fg*. No rule affects the position in the unary predicate f, thus all affected variables in R'^{2}_1 are guarded by the atom with predicate same, hence $\tau(\mathcal{R})$ is *wfg*.

Property 15 *For each fact* q *built on the initial language, if* $\tau(\mathcal{K}) \models q$ *then* $\mathcal{K} \models q$. *Moreover, if* $\mathcal{R}$ *is* gbts*, then the reciprocal holds,* i.e.*,* $\tau(\mathcal{K})$ *and* $\mathcal{K}$ *are equivalent (w.r.t. the initial language).*

Proof: $\Rightarrow$: Any $\tau(\mathcal{R})$-derivation $\mathcal{S}'$ from $\tau(F)$ can be turned into an $\mathcal{R}$-derivation $\mathcal{S}$ from F by simply ignoring the applications of rules from $\mathcal{R}'_1$ and replacing each application of a rule $\tau(R_i)$ by an application of the rule R_i with ignoring the atoms with predicate *same*.

Moreover, the facts respectively obtained by both derivations are equal on the initial language (i.e., when considering only the atoms with predicate in the initial language, and up to a variable renaming).
$\Leftarrow$: We assume that $\mathcal{R}$ is *gbts*. We show that any $\mathcal{R}$-derivation $\mathcal{S} = (F_0 = F), F_1, \ldots, F_k$ can be turned into a $\tau(\mathcal{R})$-derivation $\mathcal{S}' = (F'_0 = \tau(F)), \ldots, F'_1, \ldots F'_k$ that satisfies: (a) $\forall 0 \leq i \leq k$, F_i and F'_i are equal on the initial language; and, (b) $\forall 0 \leq i < k$, F'_{i+1} is obtained by applying $\tau(R_i)$ with a homomorphism π'_i that extends π_i. The proof is by induction on the length l of $\mathcal{S}$. The property is true for $l = 0$. Assume it is true for $l = n$. Consider the application of R_n with homomorphism π_n from $\text{body}(R_n)$ to F_n. We note $\text{fr}(R_n) = \{y_1 \ldots y_p\}$ such that $\text{body}(\tau(R_n))$ contains the atom $same(y_1, \ldots, y_p, \ldots)$. Since $\mathcal{R}$ is *gbts*, there is A_j, such that some variables from $\text{fr}(R_n)$, say $y_{i_1} \ldots y_{i_m}$ are mapped to $\text{vars}(A_j)$, and the remaining variables from $\text{fr}(R_n)$, say $y_{i_{m+1}} \ldots y_{i_p}$ are mapped to $\text{vars}F \cup \mathcal{C}$. The application of $\tau(R_j)$ in $\mathcal{S}'$ has produced a *same* atom s_1 that contains $\pi_n(y_{i_1}) \ldots \pi_n(y_{i_m})$ [by induction hypothesis (b)]. By applying Rules ${R'_1}^{3i}$ and Rule ${R'_1}^4$, we permute, and duplicate if needed (i.e., if some $y_{i_1} \ldots y_{i_m}$ have the same image by π), the arguments in s_1 to obtain the atom $s_2 = same(\pi_n(y_{i_1}), \ldots, \pi_n(y_{i_m}), \ldots)$. Then, with Rule ${R'_1}^2$, we add each $\pi_n(y_{i_j})$ for $m < j \leq p$ (note that F'_n necessarily contains $f(\pi_n(y_{i_j}))$) and build the atom $s_3 = same(\pi_n(y_{i_{m+1}}), \ldots, \pi_n(y_{i_p}), \pi_n(y_{i_1}) \ldots \pi_n(y_{i_m}), \ldots)$. Finally, with Rules ${R'_1}^{3i}$, we permute the p first arguments in s_3 to obtain $s_4 = same(\pi_n(y_1), \ldots, \pi_n(y_p), \ldots)$. Since F_n and F'_n are equal on the initial language by induction hypothesis (a), the fact obtained from F'_n after application of the previous rules from $\mathcal{R}'_2$ is still equal to F_n on the initial language. We build π'_n by extending π_n such that the atom with predicate *same* in $\text{body}(\tau(R_n))$ is mapped to s_4. Parts (a) and (b) of the induction property are thus satisfied for $l = n + 1$. □

We have just presented a polynomial reduction from the query entailment problem under *gbts* rules to query entailment under *wfg* rules. Since *wfg* is a special instance of *gbts*, it proves that the complexity of the problem for *gbts* is the same as for *wfg*. However, there is no known algorithm for *wfg*, and it thus does not provide an algorithm for *gbts*.

3.2 An optimal algorithm for *gbts*

In this section, we present an algorithm to solve the entailment problem under a *gbts* set of rules. The key idea is to exhibit some kind of regularity in the ultimate derivation tree. To that purpose, we use the notion of *bag pattern*. We first provide a procedure that allows us to maintain sound and complete bag patterns by performing a traversal of the derivation tree. Then, we perform rule applications on the pattern level. We define an equivalence relation on patterns, which ensures the following intuitive property: two bags with equivalent patterns are roots of "equivalent" trees. This motivates us to study the interaction between patterns: if a bag B of the ultimate derivation tree has pattern P, has it a child of pattern P'?

3.2.1 Patterned Forward chaining

This section focuses on *bag patterns*. We first show that forward chaining can be performed by considering solely the derivation tree endowed with bag patterns. Then we define *joins* on patterns in order to update them incrementally after each rule application. We last explain why patterns are interesting: they allow to formalize some kind of "regularity" in a derivation tree – that will be exploited in the next section to finitely represent potentially infinite derivation trees.

Definition 3.4 (Pattern) *A* pattern *of a bag* B *is a set of pairs* (G,π)*, where* G *is subset of a rule body and* π *is a partial mapping from terms*(G) *to terms*(B)*.* G *and* π *are possibly empty.*

For any derivation S, we obtain a *patterned derivation tree*, noted $PDT(S)$, by enriching the derivation tree $DT(S)$ assigning a pattern $P(B)$ to each bag B of $DT(S)$.

Definition 3.5 (Pattern soundness and completeness) *Let* F_k *be a fact obtained via a derivation* S *and let* B *be a bag in* $PDT(S)$*.* $P(B)$ *is said to be* sound *w.r.t.* F_k *if for all* $(G,\pi) \in P(B)$*,* π *is extendable to a homomorphism from* G *to* F_k*.* $P(B)$ *is said to be* complete *w.r.t.* F_k *(and* $\mathcal{R}$*), if for any* $R \in \mathcal{R}$*, any* $sb_R \subseteq body(R)$ *and any homomorphism* π *from* sb_R *to* F_k*,* $P(B)$ *contains* (sb_R,π')*, where* π' *is the restriction of* π *to the inverse image of terms*(B)*.* $PDT(S)$ *is said to be sound and complete w.r.t.* F_k *if for all its bags* B*,* $P(B)$ *is sound and complete w.r.t.* F_k*.*

Provided that $PDT(S)$ is sound and complete w.r.t. F_k, a rule R is applicable to F_k iff there is a bag in $PDT(S)$ whose pattern contains a pair $(\,body(R),-)$; then, the bag created by a rule application (R,π) on F_k has parent B_j in $DT(S)$ iff B_j is the bag in $PDT(S)$ at the smallest depth s.t. $P(B_j)$ contains $(\,body(R),\pi')$, with the restrictions of π' and π to $fr(R)$ being equal. Patterns are managed as follows: (1) The pattern of B_0 is the minimal sound and complete pattern with respect to F; (2) after each addition of a bag B_i, the patterns of all bags are updated to ensure the soundness and completeness with respect to F_i. It follows that we can define a *patterned* derivation, where rule applicability is checked on patterns, and the associated sound and complete patterned derivation tree, which is isomorphic to the derivation tree associated with the (regular) derivation.

Remember that our final goal is to avoid building the current derived fact. We will now incrementally maintain sound and complete patterns by a propagation mechanism on patterns. This is where we need to consider patterns with subsets of rule bodies and not just full rule bodies. We recall that the rules have pairwise disjoint sets of variables.

Definition 3.6 (Elementary Join) *Let* B_1 *and* B_2 *be two bags,* $e_1 = (sb^1_R,\pi_1) \in P(B_1)$ *and* $e_2 = (sb^2_R,\pi_2) \in P(B_2)$ *where* sb^1_R *and* sb^2_R *are subsets of body*(R) *for some rule* R*. Let* $V =$ *vars*$(sb^1_R) \cap$ *vars*(sb^2_R)*. The (elementary) join of* e_1 *with* e_2*, noted* $J(e_1,e_2)$*, is defined iff for all* $x \in V$*,* $\pi_1(x)$ *and* $\pi_2(x)$ *are both defined and* $\pi_1(x) = \pi_2(x)$*. Then* $J(e_1,e_2) = (sb_R,\pi)$*, where* $sb_R = sb^1_R \cup sb^2_R$ *and* $\pi = \pi_1 \cup \pi'_2$*, where* π'_2 *is the restriction*

of π_2 to the inverse image of terms(B_1) (i.e., the domain of π'_2 is the set of terms with image in terms(B_1)).

Note that V may be empty. The elementary join is not a symmetrical operation since the range of the obtained mapping is included in terms(B_1).

Example 31 *Let us consider the bags X_1 and X_2 in Figure 3.2. Let $e_1 = (\{q_2(x_4, y_4, z_4)\}, \pi)$ be in the pattern of X_1, and $e_2 = (\{s(y_4, t_4), r(z_4, t_4), h(t_4)\}, \pi')$ be in the pattern of X_2, where:*
- $\pi(x_4) = t_1^1, \pi(y_4) = u_1^1, \pi(z_4) = v_1^1$
- $\pi'(y_4) = u_1^1, \pi'(z_4) = v_1^1, \pi'(t_4) = t_2^1$

The elementary join of e_1 with e_2 is $(\{q_2(x_4, y_4, z_4), s(y_4, t_4), r(z_4, t_4), h(t_4)\}, \pi)$.

Definition 3.7 (Join) *Let B_1 and B_2 be two bags with respective patterns P_1 and P_2. The join of P_1 with P_2, denoted $J(P_1, P_2)$, is the set of pairs $J(e_1, e_2)$, where $e_1 = (sb_R^1, \pi_1) \in P_1$, $e_2 = (sb_R^2, \pi_2) \in P_2$, sb_R^1 and sb_R^2 are subsets of body(R) for some rule R.*

Note that $P_1 \subseteq J(P_1, P_2)$ since each pair from P_1 can be obtained by an elementary join with $(\emptyset, \emptyset)$. Similarly, $J(P_1, P_2)$ contains all pairs (G, π) obtained from $(G, \pi_2) \in P_2$ by restricting π_2 to the inverse image of terms(B_1).

If a pattern is sound w.r.t. F_{i-1} then it is sound w.r.t. F_i. The following property follows from the definitions:

Property 16 *If P_1 and P_2 are sound w.r.t. F_i then $J(P_1, P_2)$ is sound w.r.t. F_i.*

Proof: Follows from the definitions: for all $(G, \pi) \in J(P_1, P_2)$, either $(G, \pi) \in P_1$, or is obtained by restricting an element of P_2, or is equal to $J(e_1, e_2)$ for some $e_1 = (sb_R^1, \pi_1) \in P_1$ and $e_2 = (sb_R^2, \pi_2) \in P_2$. In the latter case, let us consider two homomorphisms, h_1 and h_2 with co-domain F_i, which respectively extend π_1 and π_2. The union of h_1 and h_2 is a mapping from $terms(G)$ to F_i (remember that h_1 and h_2 are equal on the intersection of their domains). Moreover, it is a homomorphism, because every atom in G is mapped to an atom in F_i by h_1 or by h_2. □

We consider now the step from F_{i-1} to F_i in a (patterned) derivation sequence: let B_c be the created bag and B_p be its parent in $PDT(S)$.

Definition 3.8 (Initial pattern) *The initial pattern of a bag B_c is the set of pairs (G, π) s.t. G is a subset of a rule body and π is a* homomorphism *from G to atoms(B_c).*

Example 32 (Initial pattern) *Let us consider the initial pattern of X_2 in Figure 3.2. The atoms of X_2 are:*

$$\{s(u_1^1, t_2^1), r(v_1^1, t_2^1), q_3(t_2^1, u_2^1, v_2^1)\}.$$

For rules R_1, R_2, R_{p_1} *and* R_{p_2}*, no subpart of a rule body maps to the atoms of* X_2*. Thus, they do not contribute to the initial pattern of* X_2*. There is one homomorphism from the body of* R_3 *to atoms of* X_2*, and thus its initial pattern contains:*

$$(\{q_3(t_3, u_3, v_3), \pi_a),$$

where π_a *is defined by* $\pi_a(t_3) = t_2^1, \pi_a(u_3) = u_2^1, \pi_a(v_3) = v_2^1$.

As for subparts of the body of R_4*, there are three elements added to the initial pattern of* X_2*:*

- $(\{s(y_4, t_4)\}, \pi_b)$,
- $(\{r(z_4, t_4)\}, \pi_c)$,
- $(\{s(y_4, t_4), r(z_4, t_4)\}, \pi_d)$,

where $\pi_b(y_4) = \pi_d(y_4) = u_1^1$, $\pi_b(t_4) = \pi_c(t_4) = \pi_d(t_4) = t_2^1$ *and* $\pi_c(z_4) = \pi_d(z_4) = v_1^1$.

Similar elements can be added by taking subparts of the body of R_5.

Property 17 (Soundness of initial pattern of B_c w.r.t. F_i) *The initial pattern of* B_c *is* sound *with respect to* F_i.

Proof: For any (G, π) in the initial pattern of B_c, π is a homomorphism from G to $\text{atoms}(B_c) \subseteq F_i$. □

Property 18 (Completeness of $J(P(B_c), P(B_p))$ w.r.t. F_i) *Let* $P(B_c)$ *be the initial pattern of* B_c *and* B_p *be the parent of* B_c*. Assume that* $P(B_p)$ *is complete w.r.t.* F_{i-1} *and* $\mathcal{R}$*. Then* $J(P(B_c), P(B_p))$ *is complete w.r.t.* F_i.

Proof: Let π be a homomorphism from $sb_R \subseteq \text{body}(R)$ to F_i, for some rule R. We show that $J(P(B_c), P(B_p))$ contains (sb_R, π'), where π' is the restriction of π to the inverse image of $\text{terms}B_c$. Let us partition sb_R into b_{i-1}, the subset of atoms mapped by π to F_{i-1}, and b_i the other atoms from sb_R, which are necessarily mapped by π to $F_i \setminus F_{i-1}$, i.e., $\text{atoms}(B_c)$. If b_i is not empty, by definition of the initial pattern, $P(B_c)$ contains (b_i, π_c), where π_c is the restriction of π to $\text{terms}(b_i)$. If b_{i-1} is not empty, by hypothesis (completeness of $P(B_p)$ w.r.t. F_{i-1}), P_p contains (b_{i-1}, π_p), where π_p is the restriction of $\pi_{|b_{i-1}}$ to the inverse image of $\text{terms}B_p$. If b_{i-1} or b_i is empty, (sb_R, π') belongs to $J(P(B_c), P(B_p))$ (Points 1 and 2 in Def. 3.7). Otherwise, consider $J((b_i, \pi_c), (b_{i-1}, \pi_p))$: it is equal to (sb_R, π') (Point 3 in Def. 3.7). □

Property 19 (Completeness of join-based propagation) *Assume that* $PDT(S)$ *is complete w.r.t.* F_{i-1}*, and* $P(B_c)$ *is computed by* $J(P_i(B_c), P(B_p))$*, where* $P_i(B_c)$ *is the initial pattern of* B_c*. Let* $d(B)$ *denote the distance of a bag* B *to* B_c *in* $PDT(S)$*. Updating a bag* B *consists in performing* $J(P(B), P(B'))$*, where* B' *is the neighbor of* B *s.t.* $d(B') < d(B)$*. Let* $\mathcal{T}'$ *be obtained from* $PDT(S)$ *by updating all bags by increasing value of* d*. Then* $\mathcal{T}'$ *is complete w.r.t.* F_i.

Proof: From Prop. 18, we know that $P(B_c)$ is complete w.r.t. F_i. It remains to prove the following property: let $P(B)$ be updated by $J(P(B), P(B'))$; if $P(B')$ is complete w.r.t. F_i, then $J(P(B), P(B'))$ is complete w.r.t. F_i. We partition sb_R in the same way as in the proof of Property 18. If one of the subsets is empty, we are done. Otherwise, the partition allows us to select an element e_1 from $P(B)$ and an element e_2 from $P(B')$, and $J(e_1, e_2)$ is the element we want to find. The crucial point is that if π maps an atom a of sb_R to an atom b of $F_i \setminus F_{i-1}$, and b shares a term e with B, then $e \in \text{terms}(B_c)$, hence, thanks to the running intersection property of a decomposition tree, $e \in \text{terms}(B')$, thus $(e, \pi(e))$ will be propagated to $P(B)$. □

It follows that the following steps performed at each bag creation (where B_c is introduced as a child of B_p) allow to maintain the soundness and completeness of the patterned DT:

1. initialize $P(B_c)$ with its initial pattern;
2. update $P(B_c)$ with $J(P(B_c), P(B_p))$;
3. propagate: first, propagate from $P(B_c)$ to $P(B_p)$, i.e., update $P(B_p)$ by $J(P(B_p), P(B_c))$; then, for each bag B updated from a bag B', update its children B_i (for $B_i \neq B'$) by $J(P(B_i), P(B))$ and its parent B_j by $J(P(B_j), P(B))$.

3.2.2 Bag equivalence and abstract bags

We now show how bag patterns allow us to exhibit some regularity in a derivation tree. We first need some technical, but nonetheless natural definitions. We start with the notion of *fusion* of the frontier induced by a rule application: given a rule application, it summarizes which frontier terms are mapped to the same term, and if they are mapped to a term of T_0 (that is, an initial term or a constant).

Definition 3.9 (Fusion of the frontier induced by π) *Let R be a rule and V be a set of variables with $V \cap T_0 = \emptyset$. Let π be a substitution of $fr(R)$ by $T_0 \cup V$. The* fusion *of $fr(R)$ induced by π, denoted by σ_π, is the substitution of $fr(R)$ by $fr(R) \cup T_0$ such that for all variable $x \in fr(R)$, if $\pi(x) \in V$ then $\sigma_\pi(x)$ is the smallest*[2] *variable y of $fr(R)$ such that $\pi(x) = \pi(y)$; otherwise $\sigma_\pi(x) = \pi(x) \in T_0$.*

Example 33 *Let us consider $R_2 : q_2(x_2, y_2, z_2) \rightarrow s(y_2, t_2) \wedge r(z_2, t_2) \wedge q_3(t_2, u_2, v_2)$. Let π_1 defined by $\pi_1(y_2) = y_0 = \pi_1(z_2)$. The substitution of the frontier of R_2 induced by π_2 is defined by $\sigma_{\pi_1}(y_2) = y_2 = \sigma_{\pi_1}(z_2)$. Let b be a constant, and π_2 be a substitution of the frontier of R_1 defined by $\pi_2(y_2) = b = \pi_2(z_2)$. The fusion of the frontier induced by π_2 is defined by $\sigma_{\pi_2}(y_2) = \sigma_{\pi_2}(z_2) = b$. Last, if π_3 maps y_2 and z_2 to two different existential variables, then σ_{π_3} is the identity on the frontier of R_2.*

This notion of fusion is the main tool to define *structural equivalence*, which is an equivalent relation on the bags of a derivation tree.

2. We assume variables to be totally ordered.

Definition 3.10 (Structural Equivalence) *Let* B *and* B' *be two bags in the same (prefix subtree) of a DT, or in two DTs, respectively created by applications* (R,π_i) *and* (R,π_j) *of the same rule* R. B *and* B' *are structurally equivalent if the fusions of* $fr(R)$ *induced by the restrictions of* π_i *and* π_j *to* $fr(R)$ *are equal.*

We will see a bit later that structural equivalence is not the tool we need to formalize regularity in a derivation tree. However, there is already a strong similarity between structurally equivalent bags: the purpose of Definition 3.11 is to formalize it.

Definition 3.11 (Natural bijection) *Let* B *and* B' *be two structurally equivalent bags in an abstract (prefix) DT or in two abstract (prefix) DTs. The natural bijection from* $terms(B)$ *to* $terms(B')$ *(in short from* B *to* B'*), denoted* $\psi_{B\to B'}$*, is defined as follows:*

- *if* $x \in T_0, \psi_{B\to B'}(x) = x$
- *otherwise, let* $orig(x) = \{u \in vars(head(R)) | \pi_i^{safe}(u) = x\}$. *Since* B *and* B' *are structurally equivalent,* $\forall u,u' \in orig(x), \pi_j^{safe}(u) = \pi_j^{safe}(u')$. *We define* $\psi_{B\to B'}(x) = \pi_j^{safe}(u)$.

This natural bijection between structurally equivalent bags gives us a way to partially order patterns, by ensuring that the ranges of partial applications are on the same set of terms.

Definition 3.12 (Pattern inclusion / equivalence) *Let* B *and* B' *be two bags in a (prefix subtree of a) DT or in two (prefix subtrees of) DTs, with respective patterns* $P(B)$ *and* $P(B')$. *We say that* $P(B)$ *includes* $P(B')$*, denoted* $P(B') \sqsubseteq P(B)$*, if :*

- B *and* B' *are structurally equivalent,*
- $P(B)$ *contains all elements from* $P(B')$*, up to a variable renaming given by the natural bijection:* $(G,\pi') \in P(B') \Rightarrow (G, \psi_{B'\to B} \circ \pi') \in P(B)$.

We say that $P(B)$ *and* $P(B')$ *are equivalent, denoted* $P(B) \sim P(B')$*, if* $P(B') \sqsubseteq P(B)$ *and* $P(B) \sqsubseteq P(B')$. *By extension, two bags are said to be equivalent if their patterns are equivalent.*

Property 20 explains why Definition 3.12 provides us with a good notion of pattern equivalence. Indeed, it allows us to detect similar parts in the derivation tree: if two bags are equivalent, then the "same" subtrees can be developed from them.

Property 20 *Let* F *be a fact and* $\mathcal{R}$ *be a* gbts. *Let* $\mathcal{T}$ *be the ultimate derivation tree of* F *and* $\mathcal{R}$. *Let* B *and* B' *be two bags of* $\mathcal{T}$ *such that* $P(B) \sim P(B')$. *Then for any child* B_c *of* B *by* (R,π)*, there exists a child* B'_c *of* B' *by* $(R, \psi_{B\to B'} \circ \pi)$ *such that* $P(B_c) \sim P(B'_c)$.

Proof: First, let us note that for every child B_c of B created by (R,π), there exists a child B'_c of B' created by $(R, \psi_{B\to B'} \circ \pi)$, since $P(B) \sim P(B')$. Let us assume that $P(B_c) \neq P(B'_c)$. Without loss of generality, there is some (G,π) that belongs to $P(B)$ but not to $P(B'_c)$. Since B and B' have same patterns, it means that a rule has been applied below B_c that has

not been applied below B'_c. However, this is not possible, since the first such application should be applicable identically below both bags. □

An important consequence of Property 20 is that the pattern of a bag B in the ultimate derivation tree $\mathcal{T}$ of F and $\mathcal{R}$ fully determines the subtree of $\mathcal{T}$ rooted in B (up to variable renaming). In particular, let us assume that $\mathcal{T}_b$ is a prefix subtree of the ultimate derivation tree which has been obtained from the ultimate derivation tree in the following way:

- each bag of $\mathcal{T}_b$ is either blocked or unblocked;
- for each equivalence class appearing in $\mathcal{T}$, there is exactly one unblocked node of $\mathcal{T}_b$ of that class;
- if a bag is blocked in $\mathcal{T}_b$, the whole subtree rooted in B is deleted;
- if a bag is unblocked in $\mathcal{T}_b$, all children of B in $\mathcal{T}$ are kept in $\mathcal{T}_b$.

Intuitively, we can recompute the ultimate derivation tree from the blocked tree described above by using a set of copy operations. If B and B' are equivalent, and B has a child that does not have any equivalent below B', one can copy this child below B' while keeping a prefix of the ultimate derivation tree. In the remaining of this section, we formalize the notion of full blocked tree.

Definition 3.13 (Blocked Tree) *A blocked tree is a structure $(\mathcal{T}_b, \sim)$, where $\mathcal{T}_b$ is a prefix of a patterned derivation tree and $\sim$ is the equivalence relation on the bags of $\mathcal{T}_b$ s.t. for each $\sim$-class, all but one bag are said to be* blocked*; this bag is called the* representative *of its class and is the only one that may have children.*

With a blocked tree $\mathcal{T}_b$ is associated a possibly infinite set of decomposition trees obtained by copying its bags. More precisely, this set is composed of pairs $(\mathcal{T}, f)$, where $\mathcal{T}$ is a decomposition tree obtained from $\mathcal{T}_b$ and f is a mapping from the bags of $\mathcal{T}$ to the bags of $\mathcal{T}_b$ such that for any $B \in \mathcal{T}$, B and $f(B)$ are structurally equivalent. We first define the bag copy operation:

Definition 3.14 (Bag Copy) *Let B_1 and B_2 be structurally equivalent bags with natural bijection $\psi_{B_1 \to B_2}$. Let B'_1 be a child of B_1. Copying B'_1 under B_2 (according to $\psi_{B_1 \to B_2}$) consists in adding a child B'_2 to B_2, s.t. terms(B'_2) is obtained by the following bijection b, and atoms$(B'_2) = b($ atoms$(B'_1))$: for all $x \in$ terms(B'_1), if $x \in$ terms(B_1) then $b(x) = \psi_{B_1 \to B_2}(x)$, otherwise $b(x)$ is a fresh variable.*

Assume that, in the previous definition, the bag B'_1 has been created by (R, π). Then B'_2 can be seen as obtained by the fusion of fr(R) induced by the potential application of R to B_2 with the homomorphism $\psi_{B_1 \to B_2} \circ \pi$. Since the fusions of fr(R) induced by π and $\psi_{B_1 \to B_2} \circ \pi$ are equal, B'_1 and B'_2 are structurally equivalent. Furthermore, $\psi_{B'_1 \to B'_2} = b$, where b is the bijection used in the previous definition to create B'_2 by copy of B'_1.

Starting from a block tree $\mathcal{T}_b$ and using iteratively the copy operation when applicable, one can build a possibly infinite set of trees, that we denote by $G(\mathcal{T}_b)$. It contains pairs, whose first element is a tree structure, and the second is a mapping from the bags of this tree structure to the bags of $\mathcal{T}_b$, and which intuitively encodes which bags of $\mathcal{T}_b$ have been copied to create which bag of the generated tree structure.

Definition 3.15 (Set of Trees generated by a Blocked Tree) *With a blocked tree* $\mathcal{T}_b$ *is associated a set* $G(\mathcal{T}_b)$ *inductively defined as follows:*

- *The pair* $(\mathcal{T}_b[\mathtt{root}], \mathtt{identity})$*, where* $\mathcal{T}_b[\mathtt{root}]$ *is the restriction of* $\mathcal{T}_b$ *to its root, belongs to* $G(\mathcal{T}_b)$.
- *Given a pair* $(\mathcal{T}, f) \in G(\mathcal{T}_b)$*, let* B *be a bag in* $\mathcal{T}$*, and* $B' = f(B)$*; let* B'_r *be the representative of* B' *∼-class (*$B'_r \neq B'$ *if* B' *is blocked) and* B'_c *be a child of* B'_r*. If* B *has no child structurally equivalent to* B'_c*, let* $\mathcal{T}_{new}$ *be obtained from* $\mathcal{T}$ *by copying* B'_c *under* B *(according to* $\psi_{B'_r \to B}$*), which yields a new bag* B_c*. Then* $(\mathcal{T}_{new}, f \cup (B'_c, B_c))$ *belongs to* $G(\mathcal{T}_b)$.

For each pair $(\mathcal{T}, f) \in G(\mathcal{T}_b)$*,* $\mathcal{T}$ *is said to be* generated *by* $\mathcal{T}_b$ *via* f*.* $\mathcal{T}$ *is said generated by* $\mathcal{T}_b$ *if there exists* f *such that* $\mathcal{T}$ *is generated by* $\mathcal{T}_b$ *via* f.

Note that a generated decomposition tree is not necessarily a derivation tree, but it is a prefix of a derivation tree. Among blocked trees, *full blocked trees* are of particular interest. Intuitively, a blocked tree is full if it generates only relevant trees and can generate all of them. This structure is central to our algorithm: its first version basically computes a full blocked tree, and directly reads answers on it.

Definition 3.16 *A* full blocked tree $\mathcal{T}^*$ *(of* F *and* $\mathcal{R}$*) is a blocked tree satisfying the two following properties:*

- *(Soundness) If* $\mathcal{T}'$ *is generated by* $\mathcal{T}^*$*, then there is* $\mathcal{T}''$ *generated by* $\mathcal{T}^*$ *and an* $\mathcal{R}$*-derivation* S *from* F *such that* $\mathtt{atoms}(\mathcal{T}'') = atoms(DT(S))$ *(up to fresh variable renaming) and* $\mathcal{T}'$ *is a prefix subtree of* $\mathcal{T}''$.
- *(Completeness) For all* $\mathcal{R}$*-derivations from* F*,* $DT(S)$ *is generated by* $\mathcal{T}^*$.

3.2.3 Abstract patterns and computation of a full blocked tree

We now aim at computing a full blocked tree. To that purpose, we fix a representative for each structural equivalence class, as well as for each (pattern-based) equivalence class. This is the purpose of *abstract bags* and *abstract patterns*. We also need to describe how bags of a derivation tree are related to each other: *links* are introduced to that aim. Having defined these basic components, we will focus on getting structural knowledge on the ultimate derivation tree associated with a fact and a set of rules: creation rules and evolution rules will be defined. In a last step, we use these rules to compute a full blocked tree.

We start by defining abstract bags. Each abstract bag can be seen as a canonical representative of a class of the structural equivalence relation.

Definition 3.17 (Abstract bag - Frontier terms - Generated variables) *Let* R *be a rule and* σ *a fusion of* $fr(R)$*. The abstract bag associated with* (R, σ) *(notation:* $ab(R, \sigma)$*) is such that* $terms(ab(R, \sigma)) = \sigma(terms(\mathtt{head}(R))) \cup T_0$ *and* $atoms(ab(R, \sigma)) = \sigma(\mathtt{head}(R))$*. The frontier terms of* $ab(R, \sigma)$ *are the terms that are image of a frontier*

variable of R *by* σ. *Variables of terms*$(ab(R,\sigma))$ *that are not frontier terms are called* generated variables.

Please note that we extend the natural bijection between structurally equivalent bags to abstract bags (and there is exactly one abstract bag per structural equivalence class).

Example 34 (Abstract bag) *Let us consider* $R_2 : q_2(x_2,y_2,z_2) \rightarrow s(y_2,t_2) \wedge r(z_2,t_2) \wedge q_3(t_2,u_2,v_2)$, *and three fusions* σ_{π_1}, σ_{π_2} *and* σ_{π_3} *of its frontier defined by:* $\sigma_{\pi_1}(y_2) = \sigma_{\pi_1}(z_2) = y_2$, $\sigma_{\pi_2}(y_2) = \sigma_{\pi_2}(z_2) = b$ *and* σ_{π_3} *is the identity. The abstract bag associated with* (R,σ_{π_1}) *has as terms* $\{y_2,t_2,u_2,v_2\}$ *and as atoms* $\{s(y_2,t_2), r(y_2,t_2), q_3(t_2,u_2,v_2)\}$. *The abstract bag associated with* (R,σ_{π_2}) *has as terms* $\{b,t_2,u_2,v_2\}$ *and as atoms* $\{s(b,t_2), r(b,t_2), q_3(t_2,u_2,v_2)\}$. *The abstract bag associated with* σ_{π_3} *has as terms* $\{y_2,t_2,u_2,v_2,z_2\}$ *and as atoms* $\{s(y_2,t_2), r(z_2,t_2), q_3(t_2,u_2,v_2)\}$.

Since abstract bags provide us with a canonical representative for each structural equivalence class, we can now define a canonical representative for each class of equivalent patterns: abstract patterns.

Definition 3.18 (Abstract pattern) *Let* $\mathcal{R}$ *be a set of rules,* R *be a rule and* σ *be a fusion of* $fr(R)$. *An abstract pattern with support* $B = ab(R,\sigma)$ *is a set of pairs* (G,π) *where* G *is a subset of a rule body (of a rule of* $\mathcal{R}$*) and* π *is a partial mapping from terms*(G) *to terms*(B). G *and* π *are possibly empty.*

Example 35 (Abstract pattern) *Let us consider the initial pattern described in Example 32. That pattern contains the following elements:*

- $(\{q_3(t_3,u_3,v_3)\},\pi_a)$,
- $(\{s(y_4,t_4)\},\pi_b)$,
- $(\{r(z_4,t_4)\},\pi_c)$,
- $(\{s(y_4,t_4), r(z_4,t_4)\},\pi_d)$,
- $(\{s(y_5,t_5)\},\pi_e)$,
- $(\{r(z_5,t_5)\},\pi_f)$,
- $(\{s(y_5,t_5), r(z_5,t_5)\},\pi_g)$,

with: $\pi_a(t_3) = \pi_b(t_4) = \pi_c(t_4) = \pi_d(t_4) = \pi_e(t_5) = \pi_f(t_5) = \pi_g(t_5) = t_2^1$, $\pi_a(u_3) = u_2^1$, $\pi_a(v_3) = v_2^1$, $\pi_b(y_4) = \pi_d(y_4) = \pi_e(y_5) = \pi_g(y_5) = u_1^1$ *and* $\pi_c(z_4) = \pi_d(z_4) = \pi_f(z_5) = \pi_g(z_5) = v_1^1$.

The bag it is associated with has for corresponding abstract bag $ab(R_2,id)$. *Thus, the abstract pattern associated with this initial pattern contains the same elements, where mappings are modified by substituting* t_2^1 *by* t_2, u_2^1 *by* u_2, v_2^1 *by* v_2, u_1^1 *by* y_2 *and* v_1^1 *by* z_2.

Definition 3.19 (Initial abstract pattern) *Let* R *be a rule and* σ *be a fusion of the frontier of* R. *The initial abstract pattern of* $ab(R,\sigma)$ *is the set of pairs* (G,π) *such that* G *is a subset of a rule body and* π *is a homomorphism from* G *to atoms*$(ab(R,\sigma))$.

Let B_1 and B_2 be two bags of a derivation tree such that B_2 is a child of B_1. B_1 and B_2 share some terms, *i.e.*, some terms belong to both $terms(B_1)$ and $terms(B_2)$. Let us assume that B_1 is structurally equivalent to an abstract bag B and that B_2 is structurally equivalent to an abstract bag B'. If we only state that a bag equivalent to B' is a child of a bag equivalent to B, we miss some information about the above mentioned shared terms. Capturing this information is the purpose of the notion of *link*.

Definition 3.20 (Link) *Let* B *and* B' *be two abstract bags. A link of* B *with* B' *is an injective mapping* λ *from the frontier terms of* B *to the terms of* B' *such that the range of* λ *has a non-empty intersection with the generated terms of* B'.

Please note that we constraint the mapping of the frontier terms in this way because in a derivation tree, bags are linked "as high as possible".

Example 36 (Link) *Let us consider* $R_1 : q_1(x_1,y_1,z_1) \rightarrow s(y_1,t_1) \wedge r(z_1,t_1) \wedge q_2(t_1,u_1,v_1)$ *and* $R_2 : q_2(x_2,y_2,z_2) \rightarrow s(y_2,t_2) \wedge r(z_2,t_2) \wedge q_3(t_2,u_2,v_2)$*, and the two abstract bags* $ab(R_1, id)$ *and* $ab(R_2, id)$. λ *defined by* $\lambda(y_2) = u_1$ *and* $\lambda(z_2) = v_1$ *is a link of* $ab(R_2, id)$ *with* $ab(R_1, id)$.

We are also interested in the link that describes a particular situation in a derivation tree, hence the notion of *induced* link.

Definition 3.21 (Induced link) *Let* B_1 *and* B_2 *be two bags of a derivation tree such that* B_2 *is a child of* B_1. *Let* B *and* B' *be two abstract bags such that* B *is structurally equivalent to* B_1 *and* B' *is structurally equivalent to* B_2. *The link* induced *by* B_1 *and* B_2 *is the mapping* λ *of the frontier terms of* B' *to* $terms(B)$ *such that for all* y *in the frontier terms of* B_2*:*

$$\lambda \circ \psi_{B_2 \rightarrow B'}(y) = \psi_{B_1 \rightarrow B}(y).$$

We also say that B_2 *is linked to* B_1 *by* λ.

Property 20 states that the pattern of a bag *in the ultimate derivation tree* fully determines the subtree rooted at this bag. We will thus gather information relative to the structure of the ultimate derivation tree by means of "rules" whose intuition is explained by the following example. Note that these rules have nothing to do with existential rules.

Example 37 *Let us consider* $R_1 : r(x_1,y_1) \rightarrow s(x_1,y_1)$ *and* $R_2 : s(x_2,y_2) \rightarrow p(x_2)$. *Let* P *be the following pattern:* $\{(r(x,y),\{x \rightarrow a, y \rightarrow b\}\}$. *Any bag* B *of the ultimate derivation tree such that* $P \sqsubseteq P(B)$ *is also such that* $P' \sqsubseteq P(B)$*, where* $P' = \{(r(x_1,y_1),\{x_1 \rightarrow a, y_1 \rightarrow b\}), (s(x_2,y_2),\{x_2 \rightarrow a, y_2 \rightarrow b\})\}$. *Indeed, since* $P \sqsubseteq P(B)$, R_1 *is applicable by mapping* x_1 *to* a *and* y_1 *to* b*, and thus* $s(a,b)$ *can be derived. Thus* $P' \sqsubseteq P(B)$. *Let us point out that this pattern inclusion is valid in the ultimate derivation tree, but not in the derivation tree of an arbitrary derivation sequence.*

Example 37 motivates us to define *creation* and *evolution* rules, that will be our main focus in the remaining of this section. In Example 37, we exhibited a case where we can infer that if a bag of the ultimate derivation tree has a pattern larger than P, it has also a pattern larger than P'. Such information will be gathered by evolution rules, and will be denoted by $\mathsf{P} \rightarrow \mathsf{P}'$. Creation rules describe the children that a bag of a given pattern may have. In other terms, we want to know for every (relevant) pattern P, what the possible patterns of its children are, and what the induced link for each of these patterns is. Such information will be denoted by $\mathsf{P} \rightarrow \text{Parent}(\mathsf{P}, \lambda, \mathsf{P}')$, and intuitively means that in the ultimate derivation tree, every bag B of pattern P has a child B' of pattern P' such that the link induced by B and B' is λ. We can hope to derive such information because of Property 20.

In the following, we show how to derive a set of *sound* creation and evolution rules by means of Properties 21 to 26. The constructed creation and evolution rules are sound because they describe inclusion of patterns and existence of children that are actually fulfilled in the ultimate derivation tree. Since we do not want to compute all the creation rules (for instance) for all patterns P on the left hand side of the rule, we will use these properties in order to design a procedure that works in a "lazy" manner. We will start by considering as *interesting* the initial pattern of the initial fact. At each step, we compute creation rules having an interesting pattern on the left hand side, and create all the evolution rules that can be inferred by Properties 21 to 26. Patterns appearing in the right hand side of a rule are then considered interesting, and we repeat the procedure until a fixpoint is reached.

Definition 3.22 (Creation rule - Soundness) *Let P, P' be two abstract patterns, and λ be a link between the support of P' and the support of P. A* creation rule *is a rule of the following form:*

$$\gamma_c : \mathsf{P} \rightarrow \mathit{Parent}(\mathsf{P}, \lambda, \mathsf{P}').$$

γ_c is sound *if it holds that for any bag B of the ultimate derivation tree of pattern $\mathsf{P}(\mathsf{B})$ such that $\mathsf{P} \sqsubseteq \mathsf{P}(\mathsf{B})$, there exists a child B' of B such that B' is linked by λ to B, and $\mathsf{P}' \sqsubseteq \mathsf{P}(\mathsf{B}')$.*

Knowing such creation rules would be enough to build a full blocked tree granted that we know the pattern of the root in the ultimate derivation tree. This is however not the case, and to compute it, we will rely on *evolution rules*. Such rules intuitively state the following: if the pattern of a bag in the ultimate derivation tree is greater than P, it is also greater than P'.

Definition 3.23 (Evolution rule - soundness) *Let P, P' be two abstract patterns. An* evolution rule *is a rule of the following form:*

$$\gamma_e : \mathsf{P} \rightarrow \mathsf{P}'.$$

γ_e is sound *if $\mathsf{P} \sqsubseteq \mathsf{P}'$ and for any bag B of the ultimate derivation tree such that $\mathsf{P} \sqsubseteq \mathsf{P}(\mathsf{B})$, it holds that $\mathsf{P}' \sqsubseteq \mathsf{P}(\mathsf{B})$.*

We now exhibit properties allowing to define sound rules.

Property 21 *Let* P *be a pattern,* R *be a rule,* π *be a mapping of* $fr(R)$ *to* $terms(B)$ *such that its range has a non empty intersection with the terms generated in* P*. Let us assume that* $(\ body(R), \pi)$ *is an element of* P*. Let* σ *be the fusion of* $fr(R)$ *induced by* π*. Then* $P \rightarrow\ Parent(P, \lambda, P^i_R)$ *is a sound creation rule, where:*

- P^i_R *is the initial abstract pattern of* $ab(R, \sigma)$*;*
- λ *is equal to* π *restricted to the frontier terms of* P^i_R*.*

Proof: Since the range of π has a non-empty intersection with the set of generated terms of $ab(R, \sigma)$, λ is a link of the support of P^i_R with that of P. Moreover, let B be a bag of an ultimate derivation tree such that $P \sqsubseteq P(B)$. Then $(\ body(R), \psi_{\ support(P) \rightarrow B} \circ \pi) \in P(B)$. Thus, R is applicable, by mapping its frontier in $terms(B)$ (and at least one term generated in B is image of an element of the frontier). Thus B has a child of link λ and of pattern that is bigger than P^i_R. □

We now define rules that aim at expressing that if two abstract patterns are linked in a given way, they can be enhanced by an operation similar to the join previously defined. However, the relationships between terms of different abstract patterns cannot be checked by equality as it was done when defining the join operation. We thus define abstract elementary joins, where these relationships are specified by the link between two abstract patterns. A link between two patterns is not symmetric: we thus define two join operations, to update either the abstract pattern that is the domain of the link or the abstract pattern that is the range of the link.

Definition 3.24 (Elementary abstract upper join) *Let* P_1 *and* P_2 *be two abstract patterns, and* λ *a link of* P_2 *with* P_1*. Let* $e_1 = (sb^1_R, \pi_1) \in P_1$ *and* $e_2 = (sb^2_R, \pi_2) \in P_2$*. Let* $V =$ *vars*$(sb^1_R) \cap$ *vars*(sb^2_R)*. The elementary upper join of* e_1 *with* e_2 *is defined if for all* x *in* V*,* $\pi_1(x)$ *and* $\lambda(\pi_2(x))$ *are defined and equal, and is then defined by* (sb_R, π) *where:*

- $sb_R = sb^1_R \cup sb^2_R$*,*
- $\pi = \pi_1 \cup \lambda \circ \pi'_2$*, where* π'_2 *is the restriction of* π_2 *to* $\pi_2^{-1}(\ domain(\lambda))$*.*

The elementary abstract lower join is similar, except for the definition of π: its range should be included in the terms of P_2.

Definition 3.25 (Elementary abstract lower join) *Let* P_1 *and* P_2 *be two abstract patterns, and* λ *a link of* P_2 *with* P_1*. Let* $e_1 = (sb^1_R, \pi_1) \in P_1$ *and* $e_2 = (sb^2_R, \pi_2) \in P_2$*. Let* $V =$ *vars*$(sb^1_R) \cap$ *vars*(sb^2_R)*. The elementary abstract lower join of* e_1 *with* e_2 *is defined if for all* x *in* V*,* $\pi_1(x)$ *and* $\lambda(\pi_2(x))$ *are defined and equal, and is then defined by* (sb_R, π) *where:*

- $sb_R = sb^1_R \cup sb^2_R$*,*
- $\pi = \pi_2 \cup \lambda^{-1} \circ \pi'_1$ *where* π'_1 *is the restriction of* π_1 *to* $\pi_1^{-1}(\ range(\lambda))$*.*

Abstract upper and lower joins are then defined similarly to Definition 3.7 (upper and lower join).

Definition 3.26 (Abstract upper/lower join) *Let* P_1 *and* P_2 *be two abstract patterns,* λ *a link of* P_2 *with* P_1. *The abstract upper (resp. lower) join of* P_1 *(resp.* P_2*) w.r.t.* (λ, P_2) *(resp.* (λ, P_1)*), is the set of elementary abstract upper (resp. lower) joins of* $e_1 = (sb^1_R, \pi_1) \in P_1$ *with* $e_2 = (sb^2_R, \pi_2)$, *where* sb^1_R *and* sb^2_R *are subsets of* $body(R)$ *for some rule* R. *It is denoted by* $Join_u(P_1, \lambda, P_2)$ *(resp.* $Join_l(P_1, \lambda, P_2)$*).*

We now exploit this notion of join in order to define new sound creation and evolution rules.

Property 22 *If* $P \rightarrow Parent(P, \lambda, P')$ *is a sound creation rule, then* $P \rightarrow Parent(P, \lambda, Join_l(P, \lambda, P'))$ *is also a sound rule.*

Proof: Let B and B' be two bags of the ultimate derivation tree such that $P \sqsubseteq P(B)$, $P' \sqsubseteq P(B')$, and B' is a child of B by link λ. By soundness of join propagation, $\text{Join}(P(B'), P(B)) \sqsubseteq P(B')$. By monotonicity of the join operation, it yields that $P \rightarrow \text{Parent}(P, \lambda, \text{Join}_l(P, \lambda, P'))$ is a sound rule. □

Property 23 *If* $P \rightarrow Parent(P, \lambda, P')$ *is a sound creation rule, then* $P \rightarrow Join_u(P, \lambda, P')$ *is a sound evolution rule.*

Proof: Similar to the proof of Prop 22. □

Property 24 *If* $P \rightarrow P'$ *and* $P' \rightarrow P''$ *are sound evolution rules, then* $P \rightarrow P''$ *is also a sound evolution rule.*

Proof: Let B a bag of the ultimate derivation tree such that $P \sqsubseteq P(B)$. Since $P \rightarrow P'$ is sound, then $P' \sqsubseteq P(B)$. Since $P' \rightarrow P''$ is sound, then $P'' \sqsubseteq P(B)$. Thus $P \rightarrow P''$ is sound. □

Property 25 *If* $P \rightarrow P'$ *and* $P \rightarrow Parent(P, \lambda, P'')$ *are sound evolution/creation rules, then* $P' \rightarrow Parent(P', \lambda, P'')$ *is a sound creation rule.*

Proof: It holds by monotonicity of the join operation, and by the condition that $P \rightarrow P'$ is sound implies that $P \sqsubseteq P'$. □

Property 26 *If* $P \rightarrow Parent(P, \lambda, P')$ *and* $P' \rightarrow P''$ *are sound creation/evolution rules, then* $P \rightarrow Parent(P, \lambda, P'')$ *is a sound creation rule.*

We call *pattern saturation* the already outlined procedure that builds all creation and evolution rules, and that we recall here. We define as *interesting* the initial pattern of the initial fact. Then, we start a loop that proceeds as follows. For each interesting pattern P, we use Properties 21 and 22 to build creation rules having P as left hand side. By using Properties 23, 24, 25 and 26, we saturate the set of creation and evolution rules. We then update the set of interesting patterns by inserting any pattern that appears in a right hand side of a rule. Let us prove that this process terminates.

Property 27 (Termination) *For any fact* F *and any* gbts *set of rules* $\mathcal{R}$*, pattern saturation terminates.*

Proof: There is a finite number of abstract patterns, and thus a finite number of evolution and creation rules. At each step, the number of created rules can only increase, which shows the termination of pattern saturation. □

In this fixpoint, some rules are redundant. For instance, if there exist two rules $P \rightarrow \text{Parent}(P, \lambda, P')$ and $P \rightarrow \text{Parent}(P, \lambda, P'')$, with $P' \sqsubseteq P''$, then the first rule is implied by the second one. We define the set of *most informative* creation rules as follows: a rule $P \rightarrow \text{Parent}(P, \lambda, P')$ is most informative if there is no rule $P \rightarrow \text{Parent}(P, \lambda, P'')$ with $P' \neq P''$ and $P' \sqsubseteq P''$. We define similarly the most informative evolution rules.

High-level view of the algorithm

Let us recall the main steps of the proposed algorithm:

1. (**pattern saturation**) starting from the initial pattern of the initial fact, we compute a set of evolution and creation rules that gives information on the structure of the canonical model;
2. (**full blocked tree creation**) from this set of evolution and creation rules, we build a full blocked tree – we detail this operation hereafter;
3. (**querying step**) we query the full blocked tree. If the query q is considered as a rule of the form $\text{body}(q) \rightarrow \text{match}$, where match is a new 0-ary predicate, this step can be performed by simply checking if an element (q, π) with an arbitrary π belongs to one of the patterns. If we remove the query from the patterns, we have to modify this querying operation, which will be explained in the next section.

Details on the full blocked tree computation

We now specify how to perform the second step of the algorithm. We start from the initial abstract pattern P_F associated with F. We associate it with a bag having as terms the terms of F (as well as constants appearing in rule heads). We find the most informative evolution rule having P_F as left-hand side, $P_F \rightarrow P_F^*$, and apply it. That is, we modify the pattern of the root, from P_F to P_F^*. Then, we apply every most informative creation rule having P_F^* as left member, and create corresponding bags. To apply a creation rule $P \rightarrow \text{Parent}(P, \lambda, P')$, we add a child of pattern P' to the considered bag of pattern P, such that the induced link of that child with its parent is λ. We repeat this step with the newly created pattern, adding children to at most one bag of a given pattern. This procedure halts, since there is a finite number of patterns, and the arity of the built tree is also bounded. It creates a sound blocked tree, since every creation and evolution rules are sound. It also creates a full blocked tree. Indeed, we show by induction on the length of a derivation that any patterned derivation tree obtained by a derivation of length n can be generated thanks to rules that belong to the set of rules built by the pattern saturation procedure. This is trivially true for a derivation of length 0. For a derivation of length

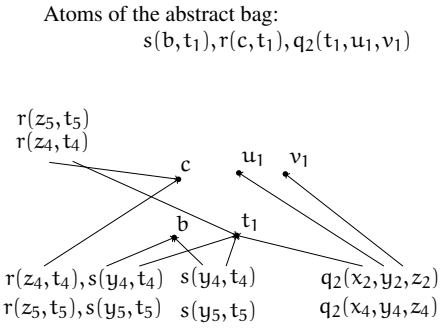

Figure 3.4: A graphic representation of $P_1^{b,c}$

n, let B the bag to which a child is linked at the last step. The derivation tree obtained before can be obtained by means of creation and evolution rules, by induction assumption. Property 21 ensures that a creation rule can create a child to B, and the saturation of the rule set by other properties ensures that the created bag has the good pattern.

We illustrate pattern saturation by expanding the running example. Writing down absolutely every element of each pattern would impede the ease of reading. We will thus allow ourselves to skip some elements, and focus on the most important ones.

Example 38 *In this example, we do not consider any query (or we may assume the query to be empty). The initial pattern* P_0 *of* F *contains the following elements:* $(q_1(x_1,y_1,z_1), \pi_1(x_1) = a, \pi_1(y_1) = b, \pi_1(z_1) = c)$, $(q_1(x_1,y_1,z_1), \pi_2(x_1) = d, \pi_2(y_1) = c, \pi_2(z_1) = e)$ *and* $(q_1(x_1,y_1,z_1), \pi_1(x_1) = f, \pi_1(y_1) = \pi_1(z_1) = g)$. *By application of Property 21, three novel rules are created:* $P_0 \to$ *Parent*$(P_0, \emptyset, P_1^{b,c})$, $P_0 \to$ *Parent*$(P_0, \emptyset, P_1^{c,e})$ *and* $P_0 \to$ *Parent*$(P_0, \emptyset, P_1^{g,g})$, *where* $P_1^{b,c}, P_1^{c,e}$ *and* $P_1^{g,g}$. *are described below.*

The abstract bag associated with $P_1^{b,c}$ *has as atoms* $s(b,t_1), r(c,t_1), q_2(t_1,u_1,v_1)$ *and an empty link (since the whole frontier of* R_1 *is mapped to constants). The abstract bag associated with* $P_1^{c,e}$ *has as atoms* $s(c,t_1), r(e,t_1), q_2(t_1,u_1,v_1)$, *and* $P_1^{g,g}$ *has as atoms* $s(g,t_1), r(g,t_1), q_2(t_1,u_1,v_1)$.

$P_1^{b,c}$ *contains the following pairs:*

- $(\{q_2(x_2,y_2,z_2)\}, (\pi_1^{b,c}(x_2) = t_1, \pi_1^{b,c}(y_2) = u_1, \pi_1^{b,c}(z_2) = v_1))$;
- $(\{q_2(x_4,y_4,z_4)\}, (\pi_2^{b,c}(x_4) = t_1, \pi_2^{b,c}(y_4) = u_1, \pi_2^{b,c}(z_4) = v_1))$;
- $(\{s(y_4,t_4)\}, (\pi_3^{b,c}(y_4) = b, \pi_3^{b,c}(t_4) = t_1))$;
- $(\{r(z_4,t_4)\}, (\pi_4^{b,c}(z_4) = c, \pi_4^{b,c}(t_4) = t_1))$;
- $(\{s(y_4,t_4), r(z_4,t_4)\}, (\pi_5^{b,c}(y_4) = b, \pi_5^{b,c}(z_4) = c, \pi_5^{b,c}(t_4) = t_1))$;
- $(\{s(y_5,t_5)\}, (\pi_6^{b,c}(y_5) = b, \pi_6^{b,c}(t_5) = t_1))$;
- $(\{r(z_5,t_5)\}, (\pi_7^{b,c}(z_5) = c, \pi_7^{b,c}(t_5) = t_1))$;
- $(\{s(y_5,t_5), r(z_5,t_5)\}, (\pi_8^{b,c}(y_5) = b, \pi_8^{b,c}(z_5) = c, \pi_8^{b,c}(t_5) = t_1))$.

$P_1^{c,e}$ *(resp.* $P_1^{g,g}$*) contains the same pairs, except that every occurrence of* b *is replaced by a* c *(resp. a* g*) and every occurrence of* c *is replaced by a* e *(resp. a* g*).* $P_1^{b,c}$ *is graphically represented in Figure 38.*

These three patterns contain $(\{q_2(x_2,y_2,z_2)\},(\pi_1^{b,c}(x_2) = t_1,\pi_1^{b,c}(y_2) = u_1,\pi_1^{b,c}(z_2) = v_1))$*, and we thus create the three following rules:*

- $P_1^{b,c} \rightarrow$ *Parent*$(P_1^{b,c},\lambda_1,P_2)$,
- $P_1^{c,e} \rightarrow$ *Parent*$(P_1^{c,e},\lambda_2,P_2)$,
- $P_1^{g,g} \rightarrow$ *Parent*$(P_1^{g,g},\lambda_3,P_2)$,

where $\lambda_1,\lambda_2,\lambda_3$ *are defined by* $\lambda_i(y_2) = u_1,\lambda_i(z_2) = v_1$ *and* P_2 *is defined below. . Note that we have to define three different* λ_i *since the supports of* $P_1^{b,c},P_1^{c,e}$ *and* $P_1^{g,g}$ *are three different abstract bags.*

P_2 *has as atoms* $\{s(y_2,t_2),r(z_2,t_2),q_3(t_2,u_2,v_2)\}$*. It contains the following elements:*

- $(\{q_3(t_3,u_3,v_3)\},(\pi_1^2(t_3) = t_2,\pi_1^2(u_3) = u_2,\pi_1^2(v_3) = v_2))$*;*
- $(\{s(y_4,t_4)\},(\pi_2^2(y_4) = y_2,\pi_2^2(t_4) = t_2))$*;*
- $(\{r(z_4,t_4)\},(\pi_3^2(z_4) = z_2,\pi_3^2(t_4) = t_2))$*;*
- $(\{s(y_4,t_4),r(z_4,t_4)\},(\pi_4^2(z_4) = z_2,\pi_4^2(y_4) = y_2,\pi_4^2(t_4) = t_2))$*;*
- $(\{s(y_5,t_5)\},(\pi_5^2(y_5) = y_2,\pi_5^2(t_5) = t_2))$*;*
- $(\{r(z_5,t_5)\},(\pi_6^2(z_5) = z_2,\pi_6^2(t_5) = t_2))$*;*
- $(\{s(y_5,t_5),r(z_5,t_5)\},(\pi_7^2(y_5) = y_2,\pi_7^2(z_5) = z_2,\pi_7^2(t_5) = t_2))$.

The element $(\{q_3(t_3,u_3,v_3)\},(\pi_1^2(t_3) = t_2,\pi_1^2(u_3) = u_2,\pi_1^2(v_3) = v_2))$ *belongs to* P_2*, and thus, we create a rule* $P_2 \rightarrow$ *Parent*(P_2,λ_4,P_3)*, where* $\lambda_3(t_3) = t_2$ *and* P_3 *contains the following elements:*

- $(\{h(t_4)\},(\pi_1^3(t_4) = t_2))$,
- $(\{h(t_5)\},(\pi_2^3(t_5) = t_2))$.

At this point, we cannot create any new rule thanks to Property 21. However, Property 23 may be used to derive an evolution of P_2*. Indeed, since* $P_2 \rightarrow$ *Parent*(P_2,λ_4,P_3) *has been derived, we can derive that* $P_2 \rightarrow P_2' =$ *Join*$_u(P_2,\lambda_4,P_3)$*;* P_2' *is a superset of* P_2 *that also contains the following elements:*

- $(\{s(y_4,t_4),h(t_4)\},(\pi_1^4(y_4) = y_2,\pi_1^4(t_4) = t_2))$*;*
- $(\{r(z_4,t_4),h(t_4)\},(\pi_2^4(z_4) = z_2,\pi_2^4(t_4) = t_2))$*;*
- $(\{s(y_4,t_4),r(z_4,t_4),h(t_4)\},(\pi_3^4(z_4) = z_2,\pi_3^4(y_4) = y_2,\pi_3^4(t_4) = t_2))$*;*
- $(\{s(y_5,t_5),h(t_5)\},(\pi_4^4(y_5) = y_2,\pi_4^4(t_5) = t_2))$*;*
- $(\{r(z_5,t_5),h(t_5)\},(\pi_5^4(z_5) = z_2,\pi_5^4(t_5) = t_2))$*;*
- $(\{s(y_5,t_5),r(z_5,t_5),h(t_5)\},(\pi_6^4(y_5) = y_2,\pi_6^4(z_5) = z_2,\pi_6^4(t_5) = t_2))$*;*
- $(\{h(t_4)\},(\pi_7^4(t_4) = t_2))$*;*
- $(\{h(t_5)\},(\pi_8^4(t_5) = t_2))$.

By Property 26, the following rules are sound:

- $P_1^{b,c} \rightarrow$ *Parent*$(P_1^{b,c},\lambda_1,P_2')$;
- $P_1^{c,e} \rightarrow$ *Parent*$(P_1^{c,e},\lambda_2,P_2')$;
- $P_1^{g,g} \rightarrow$ *Parent*$(P_1^{g,g},\lambda_3,P_2')$.

Applying once more Property 23, we get new sound rules such as:

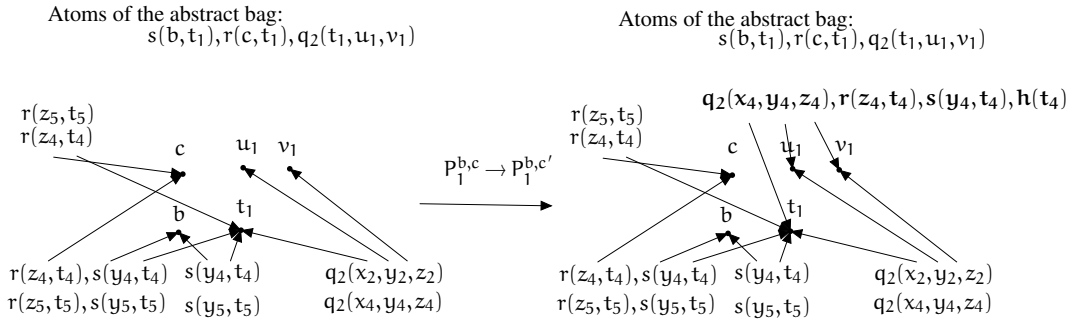

Figure 3.5: Graphical representation of the rule $P_1^{b,c} \rightarrow P_1^{b,c'}$. The new element of $P_1^{b,c'}$ is in bold.

$$P_1^{b,c} \rightarrow P_1^{b,c'},$$

where $P_1^{b,c'}$ *is a superset of* $P_1^{b,c}$ *that also contains, among others, the following element:*

$$(\{q_2(x_4,y_4,z_4), r(z_4,t_4), s(y_4,t_4), h(t_4)\}, (\pi_1^{b,c'}(x_4) = t_1, \pi_1^{b,c'}(y_4) = u_1, \pi_1^{b,c'}(z_4) = v_1)).$$

Please note that in this case, $\pi_1^{b,c'}$ *does not map every variable appearing in the corresponding subset of a rule body. Indeed,* t_4 *is not mapped, since its image by the homomorphism extending* $\pi_1^{b,c'}$ *does not belong to the terms relevant to the supporting bag.*

We skip part of the development of this example. It can be checked that at some point, a rule $P_0 \rightarrow P_0'$ *is created, where* P_0' *contains the following elements:*

$$(\{p_1(x_p), i(x_p)\}, \pi_p^1(x_p) = g) \qquad (\{p_2(x_q), i(x_q)\}, \pi_q^1(x_q) = g)$$

The following two creation rules are thus sound and relevant:

$$P_0' \rightarrow \mathit{Parent}(P_0', \emptyset, P_p^i) \qquad P_0' \rightarrow \mathit{Parent}(P_0', \emptyset, P_q^i)$$

where P_p^i *contains in particular* $(\{p_2(x_q), i(x_q)\}, \pi_q^1(x_q) = y_p)$ *and* P_q^i *contains* $(\{p_1(x_p), i(x_p)\}, \pi_p^1(x_p) = y_q)$. *Since the body of* R_{p_2} *(resp.* R_{p_1}*) belongs to* P_p^i *(resp.* P_q^i*), a new creation rule is added* $P_p^i \rightarrow \mathit{Parent}(P_p^i, \lambda_q, P_q)$ *(resp.* $P_q^i \rightarrow$

Parent(P^i_q, λ_p, P_p)*), where* $\lambda_p(x_q) = y_p$ *and* $\lambda_q(x_p) = y_q$*. Last, two recursive rules are added:*

$$P_p \rightarrow \textit{Parent}(P_p, \lambda'_q, P_q), \qquad P_q \rightarrow \textit{Parent}(P_q, \lambda'_p, P_p),$$

where $\lambda'_q(x_p) = y_q$ *and* $\lambda'_p(x_q) = y_p$.

Before turning on to the more involved querying operation, let us stress out that this first algorithm already provides a tight upper-bound for the combined complexity of query answering under *gbts* rules. Indeed, the problem is already known to be 2EXPTIME-hard, since guarded rules are a particular case of *gbts* rules.

Theorem 5 *CQ entailment for* gbts *is in* 2EXPTIME *for combined complexity and in* EXPTIME *for data complexity.*

Proof: The computation of the full blocked tree is polynomial in the size of the computed creation/evolution rule set. The number of such rules is polynomial in the number of patterns and in the maximum arity of a derivation tree. The number of patterns is double exponential in the data in the worst-case, while the arity is at most exponential. When the rule set (including the query) is fixed, the data complexity falls to EXPTIME. Lower bounds come from already known complexity of weakly-guarded rules, for instance. See the first part of the complexity analysis at the end of Section 3.3.3 for more details. □

The algorithm we proposed is thus worst-case optimal both for combined and data complexities.

3.3 Offline and online separation

We considered in previous sections the query to be a rule. This trick allowed us to have a conceptually easy querying operation, where is was sufficient to check if some bag of the full blocked tree was labeled by the query and an arbitrary mapping. However, this comes with two drawbacks. The first one is that the query is needed at the time of the construction of the full blocked tree. In scenarios where different queries are evaluated against the same data, one would like to process data and rules independently from the query, and then to evaluate the query on the pre-processed structure. This is not possible if we consider the query to be a rule. The second drawback of taking the query into account while building the full blocked tree it that it prevents us to adapt this construction when assumptions are made on the set of rules: can we devise a better algorithm if we have additional knowledge on the rule set? For instance, if it is guarded, and not only *gbts*?

This section is devoted to these issues. In the construction of the full blocked tree, we do not consider the query anymore. We still obtain a finite representation of the canonical model of F and $\mathcal{R}$, but we cannot just check if a bag is labeled by the query – since the query does not appear in any label anymore. A simple homomorphism check is not

sufficient either, as can be seen with Example 39 below. To overcome this problem, we introduce a structure called *atom-term partition tree* (APT). Such a structure is meant to encode the structure of the query induced by a homomorphism from that query to a derivation tree. A possible algorithm to check the existence of a homomorphism from a query q to a derivation tree would be to check if one of the APTs of q is the structure induced by some homomorphism π, *i.e.* to *validate* this APT. APTs and their validation in a derivation tree will be formalized in section 3.3.1. We are well aware that this definition is more involved than the simple definition of homomorphism. However, our goal will be to validate APTs, not in the potentially ever-growing derivation trees, but in the *finite* full blocked tree. In that case, APTs will still be used, but we will have to update their validation process (section 3.3.2).

Let us first stress why the usual homomorphism check is not a complete querying operation. To ease the presentation, we will restrict the running example in the following way: we only consider rules $R_{p_1} : p_1(x_p) \wedge i(x_p) \rightarrow r(x_p, y_p) \wedge p_2(y_p) \wedge i(y_p)$ and $R_{p_2} : p_2(x_q) \wedge i(x_q) \rightarrow s(x_q, y_q) \wedge p_1(y_q) \wedge i(y_q)$ (this set will be denoted by $\mathcal{R}'_r$), and the initial fact is restricted to $i(c) \wedge p_1(c) \wedge p_2(c)$ (denoted by F'_r).

Example 39 *Let us consider the following query* q_i*:*

$$q_i : p_i(x) \wedge s(x,y) \wedge r(y,z) \wedge s(z,t) \wedge r(t,u) \wedge r(x,v).$$

If we only look for a homomorphism with atoms belonging to the full blocked tree associated with $\mathcal{R}'_r$ *and* $\mathcal{F}'_r$ *and pictured in Figure 3.6, we do not find any answer to this query. However,* X_2 *is equivalent to* X_6*, and by considering a derivation tree where* X_3 *would have a corresponding bag below* X_6 *(as* X_7 *in Figure 3.7), one would find a (correct) mapping of* q_i*.*

3.3.1 Validation of an APT in a derivation tree

Let π be a homomorphism from q to the atoms of some derivation tree $T = DT(S)$. From π, let us build an arbitrary mapping π^a_T (out of the many possible ones), defined as follows: for every atom $a = p(t_1, \ldots, t_k)$ of q, let us choose a bag B of T with $a' = p(\pi(t_1), \ldots, \pi(t_k)) \in B$, and define $\pi^a_T(a) = (B, a')$. Then an *atom bag* of q (according to T and π^a_T) is a subset of atoms of q such that a and b are in the same atom bag if and only if there exists a bag B of T with $\pi^a_T(a) = (B, a')$ and $\pi^a_T(b) = (B, b')$.

Now, it is important (or it will be important in the next section) to also keep track, for a term t of q, of the bag of T in which the term $\pi(t)$ appeared. We note $\pi^t_T(t) = (B, \pi(t))$ when the term $\pi(t)$ appears in the bag B of T. We can now define a *term bag* of q in the same way as for atom bags. A *term bag* of q (according to T and π) is a subset of terms of q such that two terms u and v are in the same term-bag if and only if there exists a bag B with $\pi^t_T(u) = (B, u')$ and $\pi^t_T(v) = (B, v')$.

Then π^T_a defines, for each bag B (term or atom) of q a unique bag $\pi_T(B)$ of T: the bag where all the elements of B are mapped. More formally, if B is an atom bag of q, we have, for every atom $a \in B$, $\pi^a_T(a) = (\pi_T(B), a')$. In the same way, for every term bag B of q,

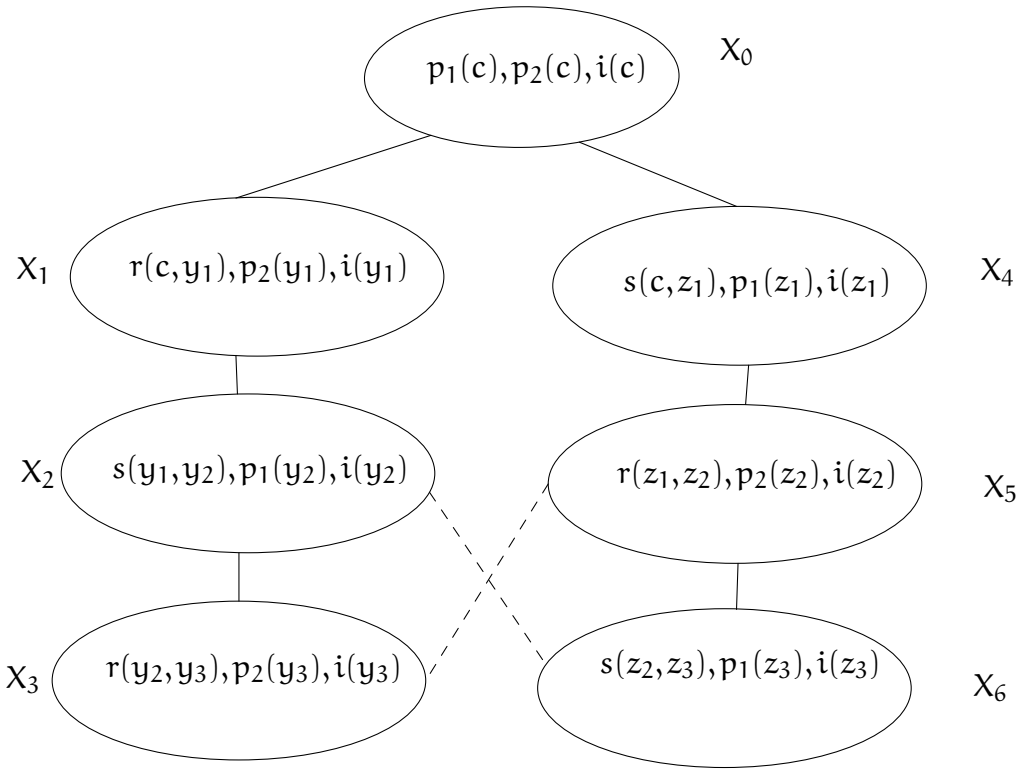

Figure 3.6: The full blocked tree associated with F'_r and $\mathcal{R}'_r$. X_2 and X_6 are equivalent, as well as X_3 and X_5.

we have, for every term $u \in B$, $\pi^t_T(u) = (\pi_T(B), u')$. We can now define the *atom-term* bags of q (induced by π^a_T). If an atom bag B_a and a term bag B_t have the same image by π_T, we obtain an atom-term bag $B = B_a \cup B_t$. If an atom bag (resp. a term bag) B have a different image than the image of any other term bag (resp. atom bag) of q, then B is an atom-term bag.

Finally, we provide these atom-term bags with a tree structure induced by the tree structure of T. Let B and B' be two atom-term bags of q. Then B' is a child of B iff *(i)* $\pi_T(B')$ is a descendant of of $\pi_T(B)$ and *(ii)* there is no atom-term bag B'' of q such that $\pi_T(B'')$ is a descendant of of $\pi_T(B)$ and $\pi_T(B')$ is a descendant of of $\pi_T(B'')$. Note that since we only consider connected queries, the structure so created is indeed a tree (it could be a forest with disconnected queries). In what follows, we define an atom-term tree decomposition of a query by such a tree of atom-term bags, independently from T and π^a_T.

Definition 3.27 *[APT of a query] Let* q *be a query. An atom-term partition of* q *is a*

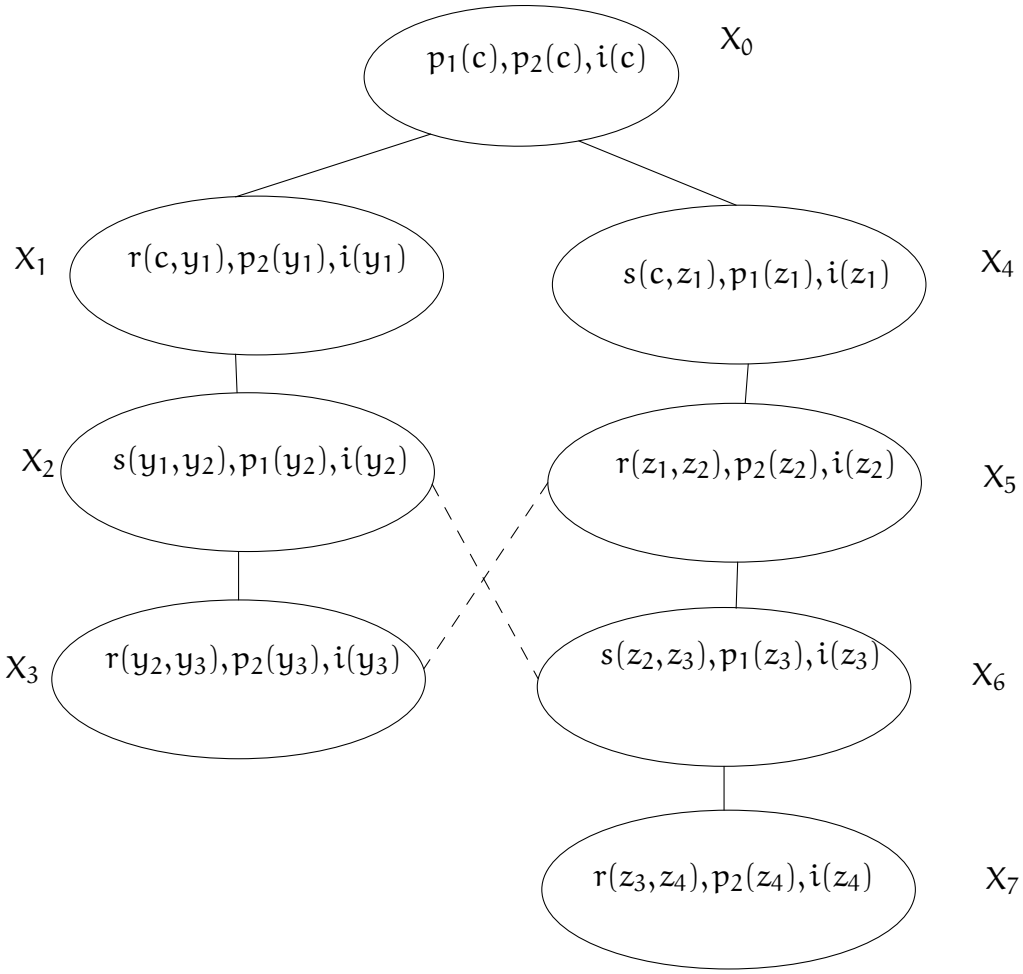

Figure 3.7: A tree generated by the full blocked tree of Figure 3.6. X_7 is a copy of X_3 under X_6.

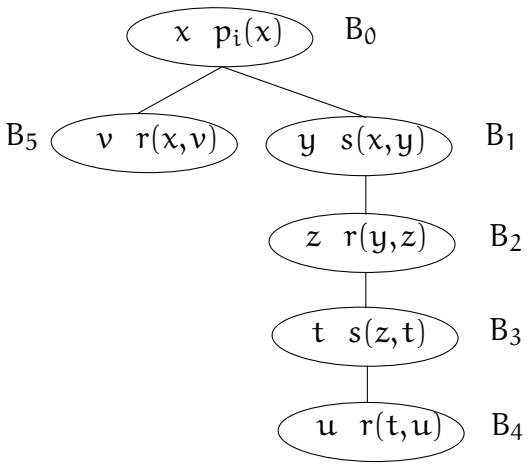

Figure 3.8: An atom-term partition of $q_i : p(x) \wedge s(x,y) \wedge r(y,z) \wedge s(z,t) \wedge r(t,u) \wedge r(x,v)$

partition of atoms and terms of q *(these sets are called* atom-term bags*). An* atom-term partition tree *(APT) of* q *is a tree whose nodes form an atom-term partition of* q.

Figure 3.8 represents an APT of the example query q_i.

Definition 3.28 *[(Valid) APT-Mapping] Let* q_t *be an APT of* q. *Let* T *be a derivation tree. An* APT-mapping *of* q_t *to* T *is a tuple* $(\Pi, \pi_1, \ldots, \pi_k)$ *where* Π *is an injective mapping from the atom-term bags of* q_t *to the bags of* T *and, for each atom-term bag* B_i *of* q_t, π_i *is a substitution from the terms of* B_i [3] *to the terms that were created in the atoms of* $\Pi(B_i)$.

Remark that if u *is a term of* q, *then* u *appears in only one atom-term bag* B_i *of* q_t. *We can thus note* $\pi(u) = \pi_i(u)$.

Finally, we say that $(\Pi, \pi_1, \ldots, \pi_k)$ *is* valid *when* π *is a homomorphism from* q *to the atoms of* $DT(S)$.

Example 40 (APT-Mapping) *We now present a valid APT-Mapping of the APT pictured Figure 3.8 to the derivation tree represented in Figure 3.7. We define* Π *as follows:*

3. We consider here the terms of B_i, not the terms appearing in the atoms of B_i.

$$\Pi(B_0) = X_0,$$
$$\Pi(B_1) = X_4,$$
$$\Pi(B_2) = X_5,$$
$$\Pi(B_3) = X_6,$$
$$\Pi(B_4) = X_7,$$
$$\Pi(B_5) = X_1.$$

The corresponding mappings are:

$$\pi_0(x) = c,$$
$$\pi_1(y) = z_1,$$
$$\pi_1(z) = z_2,$$
$$\pi_3(t) = z_3,$$
$$\pi_4(u) = z_4,$$
$$\pi_5(v) = y_1.$$

$(\Pi, \pi_1, \pi_2, \pi_3, \pi_4, \pi_5)$ *is a valid APT-mapping of the APT from Figure 3.8 to the derivation tree from Figure 3.7.*

Property 28 (Soundness and completeness) *Let* F *be a fact,* $\mathcal{R}$ *be a set of gbts rules, and* q *be a query. Then* $F, \mathcal{R} \models q$ *if and only if there exists a derivation sequence* S *from* F *to* F_k*, an APT* t *of* q*, and an APT-mapping* M *of* q_t *to* $DT(S)$ *such that* M *is valid.*

Proof: We successively prove both directions of the equivalence.

- ($\Leftarrow$) Let us suppose that there exists a valid APT-mapping from q_t to $DT(S)$. From Definition 3.28, it follows that there is a homomorphism π from q to the atoms of $DT(S)$, *i.e.* a homomorphism π from q to F_k.
- ($\Rightarrow$) If $F, \mathcal{R} \models q$, then there is a homomorphism π from q to some F_k obtained by means of a derivation S from F. As in the construction given before Definition 3.27, we can choose some mapping π_T^q of the atoms of q, and build from this mapping an APT q_t of q. We can then build an APT-mapping $(\Pi, \pi_1, \ldots, \pi_k)$ as follows: $\Pi = \pi_T$, and for each B_i of q_t, π_i is the restriction of π to the terms of B_i. This APT-mapping is valid.

□

This rather long and unnecessarily complicated way to prove the existence of a homomorphism will now be put to good use when querying the full blocked tree, without resorting to its potentially infinite development.

3.3.2 Validation of an APT in a blocked tree

Let us assume that we have an APT of a query q that corresponds to a mapping in a derivation tree $(\mathcal{T}, f)$. Thus, each bag B_q of the APT is mapped to a bag B of $\mathcal{T}$. Intuitively, we represent this mapping on the full blocked tree by mapping B_q to B_r such that $B_r = f(B)$ (*i.e.*, B has been generated by copying B_r). We can enumerate all such mappings: the question is then to validate such a mapping, that is to check that it actually corresponds to a valid APT-mapping in a tree generated by the full blocked tree.

Definition 3.29 *[Valid ADT mapping to a blocked tree] Let* q_t *be an APT of a query* q *and* $\Gamma = (\Pi, \pi_1, \ldots, \pi_k)$ *be an APT-mapping from* q_t *to a blocked tree* T_b *(where* Π *maps this time atom-term bags of* q_t *to bags of the blocked tree). Then* Γ *is said* valid *if there exists a tree generated from* T_b $(T, f) \in G(T_b)$ *and a mapping* Ξ *from the atom-term bags of* q_t *to the bags of* (T, f) *(we say that* $((T, f), \Xi$ *is a* proof that Γ is valid*) such that:*

- *if* B *is the root of* q_t*, then* $\Xi(B) = \Pi(B)$*;*
- *if* B' *is a child of* B *in* q_t*, then* $f(\Xi(B')) = \Pi(B')$ *and* $\Xi(B')$ *is a descendant of* $\Xi(B)$*;*
- *for every bag* $B_i \in (T, f)$*, let us note* Ψ_i *the bijective mapping of the terms of* $f(B_i)$ *to* B_i *(obtained by construction of* (T, f)*). Then for every atom-term bag* B_j *in* q_t *with* $\Xi(B_j) = B_i$*, we define* $\pi'_j = \Psi_i \circ \pi_j$*. Finally, the ADT mapping* $(\Xi, \pi'_1, \ldots, \pi'_k)$ *is valid in* (T, f)*.*

Example 41 (APT-Mapping to a blocked tree) *We now present a valid APT-Mapping of the APT represented Figure 3.8 to the derivation tree represented in Figure 3.7. We define* Π *as follows:*

$$\begin{aligned}
\Pi(B_0) &= X_0,\\
\Pi(B_1) &= X_4,\\
\Pi(B_2) &= X_5,\\
\Pi(B_3) &= X_6,\\
\Pi(B_4) &= X_3,\\
\Pi(B_5) &= X_1.
\end{aligned}$$

Here, the only difference with the previous APT-mapping is the image of B_4*, which is not* X_7 *(which does not exist in the blocked tree), but* X_3*. This is reflected in the definition of the* π_i*'s:*

$$\pi_0(x) = c,$$
$$\pi_1(y) = z_1,$$
$$\pi_1(z) = z_2,$$
$$\pi_3(t) = z_3,$$
$$\pi_4(u) = y_2,$$
$$\pi_5(v) = y_1.$$

$(\Pi, \pi_1, \pi_2, \pi_3, \pi_4, \pi_5)$ *is a valid APT-mapping of the APT from Figure 3.8 to the blocked tree from Figure 3.6, as witnessed by the derivation tree of Figure 3.7, where the bag* X_7 *has been generated by* X_3.

Property 29 *[Soundness and completeness] Let* F *be a fact,* $\mathcal{R}$ *be a set of* gbts *rules, and* q *be a query. Then* $F, \mathcal{R} \models q$ *if and only if there exists an APT* $_t$ *of* q*, and a valid APT-mapping from* q_t *to the full blocked tree of* F *and* $\mathcal{R}$.

Proof: We successively prove both directions of the equivalence.

- ($\Leftarrow$) Let us suppose that there exists a valid APT-mapping from q_t to the full blocked tree T_b of F and $\mathcal{R}$. From Definition 3.29, there exists a valid APT mapping from q_t to a (T, f) generated from T_b, *i.e.*, by Definition 3.16 a valid APT-mapping from q_t to some derivation tree T having (T, f) as a prefix. We can conclude thanks to Property 28.
- ($\Rightarrow$) If $F, \mathcal{R} \models q$, then there is a homomorphism π from q to some F_k obtained by means of a derivation S from F. As in the construction given before definition 3.27, we can chose some mapping π^a_T of the atoms of q, and build from this mapping an APT q_t of q. Now, in this particular π, the root of q_t can be mapped to any bag B of the derivation tree $DT(S)$. Since B has an equivalent bag B' in the full blocked tree T_b, then there exists another homomorphism π' from q to some F'_k obtained by means of a derivation S'. Let us recompute an APT q'_t (the same one can be obtained). Then (see proof of Property 28), there is a valid APT mapping $\Gamma = (\Pi, \pi_1, \ldots, \pi_k)$ from q'_t to $DT(S')$. Since $DT(S)$ is a prefix tree of some (T, f) generated from T_b. Γ is thus a valid APT mapping from q_t to (T, f).
  Now let us define the mapping Ξ as follows: if B is the root of q_t, then $\Xi(B) = \Pi(B)$, otherwise $\Xi(B) = f(\Pi(B))$. For each term t in the atom term bag B_i of q_t such that $\Xi(B_i) = B_j$, we define $\pi'_i(t) = \psi_j^{-1} \circ \pi_i$. Let us consider the APT mapping $\Gamma' = (\Xi, \pi'_1, \ldots, \pi'_k)$ from q_t to T_b. It is immediate to check that Γ' is valid.

□

3.3.3 A bounded validation for APT-mappings

Though Property 29 brings us closer to our goal to obtain an algorithm for *gbts* deduction, there is still the need to guess the generated tree (T, f) used to validate an APT-mapping (Definition 3.29). We will now show that such a generated tree can be built in a

backtrack-free manner by an exploration of the APT of the query, then establish an upper bound for each validation step (*i.e.*, if we have validated a prefix tree q'_t of the APT q_t of q, and B' is a child of some bag B in q'_t, how do we validate $'_t \cup \{B'\}$?).

Property 30 *Let q_t be an APT of q, and Γ be a valid APT-mapping from q_t to a blocked tree T_b. Let q'_t be a prefix tree of q_t, and Γ' be the restriction of Γ to q'_t. Then Γ' is a valid APT-mapping from q'_t to T_b.*

*Consider now any proof $((T',f'),\Xi')$ that Γ' is valid (see Definition 3.29). Note that this proof $((T',f'),\Xi')$ is not necessarily a subproof of the existing proof $((T,f),\Xi)$ that Γ is valid (*i.e.*, (T',f') is not necessarily a prefix tree of (T,f) and Ξ' is not necessarily the restriction of Ξ). However, there exists a proof $((T'',f''),\Xi'')$ of Γ such that $((T',f'),\Xi')$ is a subproof of $((T'',f''),\Xi'')$.*

Proof: Let us consider a proof $((T',f'),\Xi')$ of Γ'. As shown in the proof of Property 29, this proof corresponds to a homomorphism π' of q' to (T',f'). Now consider any leaf bag B' of q'_t, that is the root of a tree in q_t. Γ and Γ' can map B' to different bags in (T',f') and (T,f). However, these bags are equivalent (according to Definition 3.12) to the same bag in T_b. So anything that can be mapped under one of these bags can be mapped in the same way under the other. In particular, the subtree rooted in B' can be mapped in the same way under the bag of (T',f'). This construction leads to a homomorphism π'' from q to some (T'',f'') that extends π'. Using again the proof of Property 29, this homomorphism can be used to build a proof $((T'',f''),\Xi'')$ of Γ, that is a superproof of $((T',f'),\Xi')$. □

This latter property provides us with a backtrack-free algorithm checking the validity of an APT-mapping. Basically, Algorithm 2 performs a traversal of the APT q_t, while verifying whether Γ "correctly joins" each bag of q_t with its already "correctly joined" parent.

Algorithm 2: VALIDATEAPT

Data: A blocked tree T_b, an APT q_t, and an APT-mapping Γ from q_t to T_b.
Result: YES if Γ is valid, NO otherwise.

```
Explored := ∅;
for i = 1 to |q_t| do
    B := some bag B of q_t s.t. either parent(B, any) ∈ Explored, or B root of q_t;
    if joins(Γ, B) then
        Explored := Explored ∪{(B, joins(Γ, B))};
    else
        return NO;
return YES;
```

It remains now to explain the procedure *joins* that checks whether a valid APT-mapping of a subtree q'_t of q_t can be extended to a child B of some bag of q'_t. Let us consider a proof $((T',f'),\Xi')$ that Γ is a valid APT-mapping of q'_t. According to Def. 3.29 and Prop. 29, it is sufficient to:

– exhibit a bag B_n that can be obtained by a *bag copy sequence* $B_1, \ldots, B_n$ where $B_1 = \Xi'(\text{parent}(B))$, B_n is a bag equivalent to $\Gamma(B)$, and for $1 < i \leq n$, B_i is obtained by a bag copy (see Definition 3.14) under B_{i-1}. Having built this bag copy sequence, we have also built the bijection Ψ from the terms of $\Gamma(B)$ to the terms of B_n.
– it remains now to check that for every term t appearing in an atom of B, t is a term that belongs to a bag B_t in the branch from the root of q_t to B, and $\Xi'(t) = \Psi(\pi(t))$ (where π is the mapping defined in the APT-mapping from the terms of B to those of $\Gamma(B)$). In that case, the call to *joins* returns $\Psi \circ \pi$, ensuring that we are able to evaluate the joins of the next bags.

Our last property (whose proof is immediate) allows us to bound the length of such a bag copy sequence. Intuitively, it asserts that we do not need to generate copy bag sequences containing two bags equivalent to the same bag of T_b and that have the same frontier.

Property 31 *Let us consider a bag copy sequence* $B_1, \ldots, B_k, \ldots, B_q, \ldots, B_n$ *that validates the join of some bag* B. *Let us suppose that* $f(B_k) = f(B_q) = B'$ *(they have been copied from the same node), and that the restriction of* $\Psi_k^{-1} \circ \Psi_q$ *to the frontier terms of* B_k *is the identity. Then* $B_1, \ldots, B_k, B_{q+1}, \ldots, B_n$ *is also a bag copy sequence that validates the join of* B.

Complexity of the algorithm

The overall algorithm deciding whether $F, \mathcal{R} \models q$ can now be sketched as follows:
– build the full blocked tree T_b of $(F, \mathcal{R})$ (note that this is done independently of the query)
– for every APT q_t of q, for every APT-mapping Γ of q_t to T_b, if ValidateAPT returns YES, return YES (and return NO at the end otherwise).

Let us fix the notations used for the complexity analysis:
– b is an upper-bound of the treewidth of the canonical derivation tree. In the general case, we can fix it to the number of initial terms, plus the number of constants appearing in any rule head, plus the maximal size of a rule head
– q is the number of terms plus the number of atoms in the query
– f is the maximum size of a frontier of a rule
– a_B is the maximum number of atoms in a rule body; , t_B (resp. t_H) is the maximum number of terms in a rule body (resp. rule head).
– $|\mathcal{R}|$ is the cardinality of the rule set

The first step is done linearly in the number of evolution and creation rules. If we denote by p the number of abstract patterns, there are at most p^2 evolution rules, and $p^2 \times b^f$ creation rules. We thus need to upper-bound the number of abstract patterns. There are at most $|\mathcal{R}| \times 2^{a_B}$ subsets of rule bodies, and b^{t_B} mappings from terms of a rule body to terms of a bag. An abstract pattern being a subset of the cartesian product of these two sets, the number of abstract patterns is upper-bounded by:

$$2^{|\mathcal{R}| \times 2^{a_B} \times b^{t_B}}.$$

The first step is thus done in double exponential time, which drops to a single exponential when the set of rules is fixed. Note that only this first step is needed in the first version of the algorithm, where the query was considered as a rule, which yields the proof of Theorem 5.

The second step can be done in $N_q \times N_\Gamma \times N_V$ where:

- N_q is the number of APTs of a query q of size $|q|$, and $N_q = \mathcal{O}(2^{|q|})$ (the number of partitions on the atoms and terms of q, times the number of trees that can be built on each of these partitions);
- N_Γ is the number of APT mappings from one APT (of size $|q|$) to the full blocked tree. The size of the full blocked tree if $\mathcal{O}(b^f p)$ and thus $N_\Gamma = \mathcal{O}(p^{|q|} b^{f|q|_L})$;
- N_V is the cost of Algorithm. 2 that evaluates the validity of the APT. It performs at most q *joins*, and each one generates at most $\mathcal{O}(b^f p \times f^f)$ bags that are not frontier-equivalent (see Property 31).

The second step of our algorithm thus operates in $\mathcal{O}(p^{|q|} b^{f|q|} + b^f p f^f)$.

The querying part is thus polynomial in p (the number of patterns), and simply exponential in q and in f. Since p is in the worst-case double exponential w.r.t. F and $\mathcal{R}$, the algorithm runs in 2ExpTime. Last, given a (nondeterministically guessed) proof of Γ, we can check in polynomial time (if $\mathcal{R}$ and F are fixed) that it is indeed a valid one, yielding:

Theorem 6 *CQ entailment for* gbts *is* NP*-complete for query complexity.*

Thereby, the lower bound comes from the well-known NP-complete query complexity of plain (i.e., rule-free) CQ entailment.

3.3.4 Adaptation to relevant subclasses

We now show how to adapt the algorithm to the subclasses of *gbts* that have smaller worst-case complexity. This is done by slightly modifying the construction of the full blocked tree, allowing its size to be simply exponential or even polynomial with respect to the relevant parameters. We consider two cases:

- (weakly) guarded rules, whose combined complexity in the bounded arity case drops to ExpTime,
- guarded, frontier-guarded and frontier-1 rules, whose data complexity drops down to PTime.

For weakly guarded rules, we change the definition of pattern. Indeed, with this kind of rules, a rule application necessarily maps all the terms of a rule body to terms occurring in a single bag. This holds since every initial term belongs to every bag, and every variable of the rule body that could map to an existentially quantified variable is argument of the guard of the body of the rule. Thus, by storing all the possible mappings of a rule body atom (instead of all partial homomorphisms of a subset of a rule body), we are able to construct any homomorphism from a rule body to the current fact. The equivalence between patterns and the blocking procedure remains unchanged. Since there are at most b^w such homomorphisms for an atom, the number of abstract patterns is bounded by 2^{b^w},

which is a simple exponential since w is bounded. The algorithm thus runs in exponential time for weakly guarded rules with bounded arity.

For frontier-guarded rules (and its subclasses), we slightly modify the construction of the decomposition tree. Indeed, in the original construction, every term of the initial fact is put in every bag of the decomposition tree. However, by putting only the constants appearing in a rule head as well as every instantiation of terms of the head of the rule creating the bag (that is, we do not put all initial terms in every bags), a correct tree decomposition would also be built, and the size of the bags (except for the root) would not be dependent of the initial fact any more. The number of patterns is then upper-bounded by $1 + 2^{|\mathcal{R}| \times 2^{a_B} \times t_H^{t_B}}$. When $\mathcal{R}$ is fixed, this number is polynomial in the data. Given that q is fixed, we get the PTIME upper-bound.

3.4 Summary, related and further work

In this chapter, we focused on sets of rules fulfilling the bounded treewidth property. More specifically, we designed a class of rules, namely the greedy bounded treewidth sets, that generalizes guarded-based rules – and as such lightweight description logics. We proposed a worst-case optimal algorithm for that class, and adapted it for known subclasses. The presented algorithm is based on the construction of a finite representation of the derivation tree of the (potentially infinite) canonical model. The blocking technique that we used is based on the notion of a pattern, which summarizes the relevant information about mappings of (parts of) body rules.

Some results combine well with those presented here. First, it has been shown that conjunctive query entailment under frontier-1 rules is 2EXPTIME-hard, which shows that the presented algorithm is also worst-case optimal for that class of rules as well. Second, we have not presented the complexity of the recognizability problem. It happens to be exactly as hard as the query answering problem. Both membership [Baget, 2012] and hardness [Rudolph, 2012] are shown by a reduction to (or from) query answering with *gbts* rules.

A lot of questions are still open for further work. First, a detailed comparison with two other algorithms would certainly be beneficial. The original algorithm for guarded rules [Calì *et al.*, 2009] is based on a notion of *type*, which has some similarities with the notion of pattern – without being exactly the same object. Another source of comparison is the family of *combined approach* algorithms [Lutz *et al.*, 2009; Kontchakov *et al.*, 2010, 2011]. In these algorithms, a finite representation of the canonical model is also built. However, this finite representation is an over-approximation of the canonical model, and queries need to be rewritten (either ontology-independently or not) in order to regain soundness of homomorphism as a querying mechanism. Studying more precisely the similarities between both algorithms may lead to an algorithm which could deal with more expressive classes than lightweight description logics while taking advantage of a query rewriting approach.

This is in particular relevant since the querying operation that we propose, though theoretically of adequate worst-case complexity, does not seem to be readily and efficiently implementable. While we believe that the representation that we used could be of practical interest in some settings, a significant amount of work is needed to design a good querying operation. A rewriting based approach is a natural candidate. To evaluate it, one could be interested in fragments of the *gbts* class for which one could use a simpler equivalence relation than the one introduced in this dissertation. As an example, one could wish to use structural equivalence. Defining a class of rules for which structural equivalence behaves well with respect to the construction of the full blocked tree is an interesting open question.

CHAPTER

4 Materialization-avoiding approach

Preamble

In this chapter, we focus on query rewriting approaches to Ontology-Based Query Answering (OBQA). We study rewritings into unions of conjunctive queries, and stress one of their limits: the large size of rewritings when large class or relation hierarchies are involved. To address that limit, we propose to use another kind of queries to perform query rewriting, that we call semi-conjunctive queries*, and present an algorithm that computes such rewritings. Last, we experimentally evaluate their* quality*, by comparing the evaluation time of such a query to the evaluation time of an equivalent union of conjunctive queries.*

Contents

While materialization-based approaches have been proven efficient in some useful cases, such techniques suffer from several drawbacks. Indeed, the very idea of materializing data has some undesirable consequences. First, one should have the right to add new data in a database, which might not always be the case, depending on the context. But even if such rights are granted, two major problems remain: the size of the data may blow-up, and data updates are not easy to deal with.

Indeed, the size of the materialized data could be exponential with respect to the size of the original fact. But even if it were of linear size, it still may be not acceptable

when dealing with facts that have billions – or more – of atoms. The other concern is about data updates. Let us assume that all the consequences derivable from a fact F and a set of existential rules $\mathcal{R}$ have been materialized – a potentially costly operation. However, for some unknown (but realistic) reason, one of the atoms of the database has to be removed. How to recompute the materialization? To the best of our knowledge, no algorithm has been proposed yet to update the materialization without recomputing it from scratch. In scenarios where updates could occur on a regular basis, such an approach is not acceptable.

Thus, a consequent research effort has been made towards querying techniques that do not require materialization, aiming at overcoming the presented drawbacks. Most of the results focus on rewriting a conjunctive query into a union of conjunctive queries (UCQ). Starting from a query q and a set of rules $\mathcal{R}$, one computes a union of conjunctive queries $\mathcal{Q}$, such that for any fact F, it holds that $F,\mathcal{R} \models q$ if and only if $F \models \mathcal{Q}$. More generally, we will consider rewritings that are not necessarily unions of conjunctive queries, but formulas of a given class Φ. We recall the definition of a Φ-rewriting (Definition 2.17).

Definition 4.1 (Φ-rewriting soundness/completeness) *Let Φ be a class of first-order formulas, $\mathcal{R}$ be a set of rules, and q be a conjunctive query. $\varphi \in \Phi$ is a* sound *rewriting of q with respect to $\mathcal{R}$ if for any fact F, it holds that $F \models \varphi$ implies that $F,\mathcal{R} \models q$. φ is a* complete *rewriting of q with respect to $\mathcal{R}$ if for any fact F, the converse holds, that is: $F,\mathcal{R} \models q$ implies that $F \models \varphi$.*

A recurrent aim of the literature about UCQ-rewritings has been to produce sound and complete rewritings that are as small as possible.

In the remaining of this chapter, we first characterize optimal UCQ-rewritings, that is, UCQ-rewritings of minimal size among sound and complete UCQ-rewritings, where the size of a UCQ is the number of conjunctive queries in it. We point out that the piece-based rewriting algorithm (Algorithm 1, Chapter 2) generates such an optimal UCQ-rewriting. After advocating that UCQs are not well suited to represent sound and complete rewritings, especially when large class or role hierarchies are present, we introduce (unions of) *semi-conjunctive queries* ((U)SCQ), which are a more general form of positive existential formulas. We propose an algorithm for computing sound and complete USCQ-rewritings. We then provide some experiments which aim at showing that USCQ-rewritings are both more efficiently generable and more efficiently evaluable than equivalent UCQ-rewritings.

We recall here, to ease the reading of this dissertation, the running example that has been used in Chapter 2 in order to illustrate the backward chaining algorithm introduced there.

Example 42 *Let $\mathcal{R}_e = \{R_1, R_2, R_3, R_4, R_5\}$, defined as follows:*

- $R_1 : p(x) \wedge h(x) \rightarrow s(x,y)$
- $R_2 : f(x) \rightarrow s(x,y)$
- $R_3 : f_1(x) \rightarrow s_1(x,y)$
- $R_4 : t(x,y) \rightarrow t(y,x)$
- $R_5 : s_1(x,y) \rightarrow s(x,y)$

Let q_e *be the following Boolean query:*

$$q_e = t(x_1, x_2) \wedge s(x_1, x_3) \wedge s(x_2, x_3),$$

We also recall some basic notations. A conjunctive query, as well as an existentially closed disjunction of atoms, will be considered as a set of atoms. A union of conjunctive queries will be considered as a set of conjunctive queries. In spite of the potential ambiguity – should a set be interpreted as a conjunction or as a disjunction of its elements? – this is a convenient notation. The context will make clear whether a set is a disjunction or a conjunction. Given a set of atoms q and $q' \subseteq q$, we call the *separator* of q' (with respect to q) the set of variables that occur both in q' and $q \setminus q'$: $\text{sep}_q(q') = \text{vars}(q') \cap \text{vars}(q \setminus q')$. More generally, given a set of sets of atoms $s = \{d_1, \ldots, d_n\}$, and $s' \subseteq s$, the separator of s' with respect to s, denoted by $\text{sep}_s(s')$, is the set of variables that appear in both s' and $s \setminus s'$.

4.1 UCQ-rewritings

The most common approach to query rewriting is to rewrite a conjunctive query into a union of conjunctive queries. This approach has been initiated in the paper defining the DL-Lite family [Calvanese *et al.*, 2007], whose design rationale is to allow such rewritings. Since then, a number of prototypes have been implemented, most of them focusing on lightweight description logics. More details are given in Section 4.4. Over a few years, there has been a dramatic reduction of the size of the computed UCQ-rewritings: in some cases, for the same query and ontology, the size of the computed UCQ-rewriting dropped from a hundred of thousands conjunctive queries to a few hundreds. This raises an interesting and very natural question: given a query q and a set of rules $\mathcal{R}$, can we characterize the smallest union of conjunctive queries that is a sound and complete rewriting of q with respect to $\mathcal{R}$?

4.1.1 Minimality of a sound and complete UCQ-rewriting

Let φ_1 and φ_2 be two first-order formulas. If φ_1 is logically entailed by φ_2, we note $\varphi_2 \models \varphi_1$ or $\varphi_1 \geq \varphi_2$. In the case of conjunctive queries, this relation can be verified thanks to a homomorphism check. As already noted in Chapter 2 for UCQ-rewritings, if q is a conjunctive query and $\mathcal{Q}$ is a Φ-rewriting of q that contains φ_1 and φ_2 such that $\varphi_1 \geq \varphi_2$, then $\mathcal{Q} \setminus \{\varphi_2\}$ is also a sound and complete rewriting of q. This observation motivated the definition of *cover* of a set of first-order queries, that we recall here.

Definition 4.2 (Cover) *Let $\mathcal{F}$ be a set of first-order queries. A* cover *of $\mathcal{F}$ is a set $\mathcal{F}^c \subseteq \mathcal{F}$ such that:*

1. *for any $f \in \mathcal{F}$, there is $f' \in \mathcal{F}^c$ such that $f' \geq f$,*
2. *elements of $\mathcal{F}^c$ are pairwise incomparable with respect to $\geq$.*

Example 43 *Let* $\mathcal{Q} = \{q_1 = r(x,y) \wedge t(y,z), q_2 = r(x,y) \wedge t(y,y), q_3 = r(x,y) \wedge t(y,z) \wedge t(u,z)\}$. *A cover of* $\mathcal{Q}$ *is* $\{q_1\}$. *Indeed,* $q_1 \geq q_2$ *and* $q_1 \geq q_3$, *because for* $i \in \{2,3\}$, $\pi_{1\to i}$ *is a homomorphism from* q_1 *to* q_i *where:*

- $\pi_{1\to 2}(x) = x, \pi_{1\to 2}(y) = \pi_{1\to 2}(z) = y$, *and*
- $\pi_{1\to 3}(x) = x, \pi_{1\to 3}(y) = y, \pi_{1\to 3}(z) = z$.

The following property is immediate:

Property 32 *Let* $\mathcal{Q}$ *be a sound and complete rewriting of* q. *A cover of* $\mathcal{Q}$ *is a sound and complete rewriting of* q. *Moreover, its cardinality is smaller than or equal to the cardinality of* $\mathcal{Q}$.

More interestingly, in the special case of conjunctive queries, any cover of any sound and complete UCQ-rewriting is of minimal size among sound and complete UCQ-rewritings. This is the topic of Theorem 7.

Theorem 7 *Let* $\mathcal{R}$ *be a set of existential rules,* q *a conjunctive query that admits a finite sound and complete UCQ-rewriting* $\mathcal{Q}$. *Any cover of* $\mathcal{Q}$ *is of minimal cardinality among sound and complete rewriting sets of* q *with respect to* $\mathcal{R}$.

Proof: Let $\mathcal{Q}_1$ and $\mathcal{Q}_2$ be two arbitrary finite sound and complete rewriting sets of q with respect to $\mathcal{R}$. Let $\mathcal{Q}_1^c$ (resp. $\mathcal{Q}_2^c$) be a cover of $\mathcal{Q}_1$ (resp. of $\mathcal{Q}_2$). $\mathcal{Q}_1^c$ and $\mathcal{Q}_2^c$ are also sound and complete, and are of smaller cardinality. We show that they have the same cardinality. Let $q_1 \in \mathcal{Q}_1^c$. There exists $q_2 \in \mathcal{Q}_2^c$ such that $q_1 \leq q_2$. If not, q would be entailed by $(F = q_1, \mathcal{R})$, since $\mathcal{Q}_1^c$ is a sound rewriting of q (and q_1 maps to itself), but not element of $\mathcal{Q}_2^c$ would map to F. This would show that $\mathcal{Q}_2^c$ is not complete, which is absurd. Similarly, there exists $q_1' \in \mathcal{Q}_1^c$ such that $q_2 \leq q_1'$. Thus, $q_1 \leq q_1'$, which implies that $q_1' = q_1$, since no two distinct elements of $\mathcal{Q}_1^c$ are comparable for $\geq$. Thus, for all $q_1 \in \mathcal{Q}_1^c$, there exists $q_2 \in \mathcal{Q}_2^c$ such that $q_1 \leq q_2$ and $q_2 \leq q_1$. Such a q_2 is unique: indeed, two such elements would be comparable for $\leq$. The function associating q_1 with q_2 is thus a bijection from $\mathcal{Q}_1^c$ to $\mathcal{Q}_2^c$, which shows that these two sets have the same cardinality. □

If we additionally restrict the queries to be isomorphic to their core, the proof of Theorem 7 yields that there exists a unique optimal sound and complete UCQ-rewriting for any query and any finite unification set.

4.1.2 Limitation of UCQ-rewritings

Since Algorithm 1 of Chapter 2 outputs a sound and complete rewriting that is its own cover, we can compute an optimal (*i.e.*, with the smallest number of conjunctive queries) sound and complete UCQ-rewriting for any query q and any finite unification set. However, this does not state anything about the practical feasibility of computing a UCQ-rewriting. Indeed, the following example shows that given a realistic ontology (composed of a simple binary relation hierarchy) and a moderately small query (two atoms), the size of the optimal UCQ-rewriting is huge.

Example 44 *Let* $\mathcal{R} = \{R_i\}_{1 \leq i \leq n}$*, where* $R_i : r_i(x,y) \rightarrow r_{i-1}(x,y)$*. Let* q *be the following query:*

$$r_0(x_1,x_2) \wedge r_0(x_2,x_3).$$

q *has* $(n+1)^2$ *rewritings, which are* $\{r_i(x_1,x_2) \wedge r_j(x_2,x_3)\}_{0 \leq i,j \leq n}$.

Please note that, as in Chapter 2, we freely renamed variables of the rewritings, which is not a problem since all these variables are mute variables. We will allow ourselves such renaming throughout this chapter.

Example 44 can be generalized by taking a query of k atoms and classes/roles having n subclasses/subroles. This would yield an optimal UCQ-rewriting of $(n+1)^k$ conjunctive queries. While RDMS are efficient at dealing with small UCQs, this raises the question whether UCQs are a good choice as a target language for rewritings of conjunctive queries, when ontologies contain large role and class hierarchies.

4.2 USCQ-rewritings: definition and computation

Theorem 7 ensures that obtaining large UCQ-rewritings is not a matter of a wrong algorithm, but a consequence of using UCQs to represent rewritings. At least three approaches can be considered:

- computing UCQs that are no longer complete, that is, that may not give all the correct answers when evaluated on some databases. Depending on the application, completeness might not be an absolute requirement;
- exploiting additional properties of the data, as made in [Rodriguez-Muro and Calvanese, 2012]. It is advocated that queries are not evaluated against arbitrary databases, but against databases that respect a given schema. These databases are thus already complete with respect to some predicates, and all rewritings may not be needed;
- using a different kind of queries to represent sound and complete rewritings. This is the approach that we adopt in the remaining of this chapter. Such an approach has already been undertaken, in particular by using Datalog queries instead of union of conjunctive queries. However, our contribution extends the applicability of such techniques.

Example 45 shows what a (restricted) use of disjunction can bring with respect to a union of conjunctive queries.

Example 45 *A formula equivalent to the UCQ-rewriting of Example 44 would be:*

$$(\vee_{i=0}^{n} r_i(x_1,x_2)) \wedge (\vee_{j=0}^{n} r_j(x_2,x_3)),$$

Although we make use of disjunction, this use is strongly limited. We call *semi-conjunctive queries* the formulas that follow these limitations.

Definition 4.3 (Semi-conjunctive query) *A* semi-conjunctive query *(SCQ) is a closed logical formula of the following form:*

$$\exists \mathbf{x}\ D_1 \wedge D_2 \wedge \ldots \wedge D_n$$

where D_i *is a disjunction of atoms (for any* i*), and* $\mathbf{x}$ is the set of variables that appear in the formula. We also note $\{D_1, \ldots, D_n\}$.

In Examples 44 and 45, the difference in representation's compactness is striking, in particular when generalizing them with k atoms. The research questions that we will tackle in the remaining of this chapter are the following:

1. given a conjunctive query q and a set of existential rules $\mathcal{R}$, can we efficiently compute a sound and complete USCQ-rewriting of q with respect to $\mathcal{R}$?
2. given a USCQ, can we efficiently evaluate this query against a database?

Please note that the second question is about the evaluation efficiency, and not the size of a query. Indeed, to the best of our knowledge, no size function on first-order formulas is known to be related with the practical evaluation efficiency of these formulas against a database. Finding such a measurement would be very interesting, but is out of the scope of this thesis.

Of course, any SCQ is equivalent to a UCQ – in particular, this is the case for the union of its *selections*.

Definition 4.4 (Selection) *Let* s *be an SCQ. A* selection *of* s *is a CQ* $q = \wedge_{d \in s}\ i(d)$*, where* i *is a function that maps each* $d \in s$ *to some* $i(d) \in d$ *(where* d *is considered as a set of atoms).*

We explicit in Example 46 the selections of the SCQs involved in Example 45.

Example 46 (Selection) *Let* $s = (r_0(x,y) \vee r_1(x,y)) \wedge (r_0(y,z) \vee r_1(y,z))$. s *has four selections, which are:*

- $r_0(x,y) \wedge r_0(y,z)$,
- $r_0(x,y) \wedge r_1(y,z)$,
- $r_1(x,y) \wedge r_0(y,z)$,
- $r_1(x,y) \wedge r_1(y,z)$.

Property 33 *Any SCQ* s *is equivalent to the disjunction of its selections.*

We introduce some convenient definitions about semi-conjunctive queries. Definition 4.5 defines shared variables for a SCQ.

Definition 4.5 (Shared variable) *Let* s *be an SCQ. A* shared variable *of* s *is a variable that is argument of two atoms* a *and* b *such that* a *and* b *belongs to two different disjunctions of* s.

Example 47 *Let* $s = (r_0(x,y) \vee r_1(x,y)) \wedge (r_0(y,z) \vee r_1(y,z))$ *be the SCQ of Example 46.* y *is a shared variable of* s*, while* x *and* z *are not.*

We also need a technical restriction on SCQs: Definition 4.6 introduces *well-formed* SCQs.

Definition 4.6 (Well-formed SCQ) *An SCQ* s *is* well-formed *if for any shared variable* x *of* s *and any disjunction* d *of* s*, either* x *does not appear in* d*, or* x *is an argument of any atom of* d*.*

Example 48 presents two SCQs: one is well-formed, and the other is not.

Example 48 $(r_0(x,y) \vee r_1(x,y)) \wedge (r_0(y,z) \vee r_1(y,z))$ *is well-formed. Indeed,* y *is the only shared variable and appears in every atom. On the other hand,* $(p(x) \vee r_1(x,y)) \wedge (r_0(y,z) \vee r_1(y,z))$ *is not well-formed:* y *is shared, appears in the disjunction* $(p(x) \vee r_1(x,y))$*, but not in the atom* $p(x)$*.*

In the remaining of this section, we will only consider well-formed SCQs, even though we will never refer again to that condition, since the algorithm we propose generates at each step only well-formed SCQs.

4.2.1 Piece-unifiers for semi-conjunctive queries

In Chapter 2, we presented a unification operation between a conjunctive query and an existential rule. We now focus on generalizing this notion to a unification between a semi-conjunctive query and a rule. We overload the term piece-unifier, since it should not cause any confusion. Moreover, we will simply call them unifier in the following.

Definition 4.7 (Piece-unifier) *Let* s *be an SCQ and* R *be a rule. A* piece-unifier *of* s *with* R *is a triple* $\mu = (s', q', u)$ *with* $s' \subseteq s$*,* $s' \neq \emptyset$*,* q' *a selection of* s'*, and* u *a substitution of* $fr(R) \cup vars(s')$ *by terms*$(head(R)) \cup \mathcal{C}$ *such that:*

1. *for all* $x \in fr(R)$*,* $u(x) \in fr(R) \cup \mathcal{C}$ *(for technical convenience, we allow* $u(x) = x$*);*
2. *for all* $x \in sep_s(s')$*,* $u(x) \in fr(R) \cup \mathcal{C}$*;*
3. $u(q') \subseteq u(head(R))$*.*

We recall that $sep_s(s') = vars(s') \cap vars(s \setminus s')$*.*

We illustrate this definition with Example 49.

Example 49 *Let* $s = t(x_1,x_2) \wedge (s_1(x_1,x_3) \vee s(x_1,x_3)) \wedge (s_1(x_2,x_3) \vee s(x_2,x_3))$*.* $\mu = (s', q', u)$ *is a unifier of* s *with* $R_3 = f_1(x) \rightarrow s_1(x,y)$ *where:*

- $s' = (s_1(x_1,x_3) \vee s(x_1,x_3)) \wedge (s_1(x_2,x_3) \vee s(x_2,x_3))$
- $q' = s_1(x_1,x_3) \wedge s_1(x_2,x_3)$
- $u(x_1) = u(x_2) = u(x) = x, u(x_3) = u(y) = y$.

Local and non-local unifiers

The major novelty of our approach is the distinction between *local* and *non-local* unifiers. Local unifiers unify a query with an atomic-body rule; furthermore, they unify only one disjunction at a time (hence, only one atom of the query) and do not merge any pair of terms of the query or specialize any variable into a constant.

Definition 4.8 (Local unifier) *Let* s *be an SCQ,* R *be an atomic-body rule and* $\mu = (s', q', u)$ *be a unifier of* s *with* R. μ *is* local *if* q' *is restricted to a single atom and if the restriction of* u *to terms*(q') *is injective and does not map a variable to a constant.*

Example 50 illustrates the notion of a local unifier.

Example 50 (Local unifier) *The unifier in Example 49 is not local, since two disjunctions are unified at once.*

Let μ_L *be the unifier of* $q_e = t(x_1, x_2) \wedge s(x_1, x_3) \wedge s(x_2, x_3)$ *with* $R_4 = t(x, y) \rightarrow t(y, x)$ *defined by:* $\mu_L = (t(x_1, x_2), t(x_1, x_2), (u(x_1) = y, u(x_2) = x))$. μ_L *is a local unifier.*

The importance of local unifiers relies on the associated rewriting operation. We perform two different kinds of rewritings: if a unifier is not local, the rewriting operation is a natural recast in the framework of SCQs of the usual operation of piece-based rewriting. However, when a unifier is local, the associated rewriting operation introduces disjunction. We now present local rewritings, but we first need the notion of X-entailment.

Definition 4.9 (X-entailment) *Let* D *be a set of atoms, and* X *a set of variables. Let* a *be an atom.* a *is* X-entailed *by* D *if there is a homomorphism* π *from* a *to* D *such that if* $x \in var(a) \cap X$*, then* $\pi(x) = x$.

X-entailment is illustrated by Example 51.

Example 51 *Let* $D = \{r(x, y), p(x, u)\}$. $p(x, v)$ *is* $\{x\}$ *entailed by* D*, but* $r(y, x)$ *is not.*

We can now focus on the fundamental definition of a *local rewriting*.

Definition 4.10 (Local rewriting) *Let* $s = \bigwedge_{i=1}^{n} d_i$ *be an SCQ,* R *be an atomic-body rule and* $\mu = (s' = \{d_1\}, q', u)$ *be a local piece-unifier of* R *with* s. *The* local rewriting *of* s *with respect to* R *and* μ *(denoted by* $\gamma_L(s, R, \mu)$*) is* $d'_1 \wedge \bigwedge_{i=2}^{n} u(d_i)$*, where* $d'_1 = u(d_1) \vee u($ *body*$(R))$*, if* $u($ *body*$(R))$ *is not* $sep_{d'_1 \wedge \bigwedge_{i=2}^{n} u(d_i)}(\{u(d_1)\})$*-entailed by* $u(d_1)$*, and* s *otherwise. We define the natural bijection* b *between disjunctions of* s *and disjunctions of* s' *by* $b(d_1) = d'_1$ *and* $b(d) = u(d)$ *otherwise.*

In Definition 4.10, the condition about $sep_{d'_1 \wedge \bigwedge_{i=2}^{n} u(d_i)}(\{u(d_1)\})$-entailment is here to ensure that "equivalent" atoms are not added, which would prevent the rewriting process to terminate.

We illustrate the notion of local rewriting in Example 52. The key point is the appearance of disjunction.

Example 52 (Local rewriting) *Let* μ_L *be the unifier of* q_e *with* R_4 *as defined in Example 50. The rewriting of* q_e *with respect to* μ_L *is* q'_e *defined by:*

$$(t(x_1,x_2) \vee t(x_2,x_1)) \wedge s(x_1,x_3) \wedge s(x_2,x_3)$$

We now present a technical property that will be useful in subsequent proofs: the set of shared variables remains unchanged after applying a local unifier.

Property 34 *Let* s *be an SCQ,* R *be an existential rule, and* $\mu = (s', q', u)$ *be a local unifier of* s *with* R*. For any* $d \in s$*, it holds that:*

$$u(\ sep_s(d)) = \ sep_{s'}(b(d)).$$

Proof: Let $x \in \ \text{sep}_s(d)$. It implies that $x \in \ \text{vars}(d)$ and thus there exists $d' \neq d$ such that $x \in \ \text{vars}(d')$. $u(x)$ belongs to both $b(d)$ and $b(d')$, and thus belongs to $\ \text{sep}_{s'}(d)$. For the converse inclusion, let us first notice that since rules and queries do not share any variable, any variable of s' that is not the image by u of a variable of s appears in the newly added atom, and thus cannot be shared. Let $y = u(x) \in \ \text{sep}_{s'}(b(d))$. There exists $d' \in s$ such that y appears also in $b(d')$. But then x appears both in d and d', and belongs to $\ \text{sep}_s(d)$. □

We finally present the definition of non-local rewriting. As for local unifiers, a subset s' of the disjunction is chosen, as well as a selection q' of s'. However, in the case of non-local rewritings, these disjunctions are removed and a set of new disjunctions are added: for each atom of the body of the rule R, a disjunction restricted to a single atom is added.

Definition 4.11 (Non-local rewriting) *Let* $s = \bigwedge_{i=1}^{n} d_i$ *be an SCQ,* R *be a rule, and* $\mu = (s' = \{d_1, \ldots, d_k\}, q', u)$ *be a non-local unifier of* R *with* s*. The* non-local rewriting *of* s *with respect to* R *and* μ *(denoted by* $\gamma_{NL}(s, R, \mu)$*) is* $u(\ body(R)) \wedge \bigwedge_{i=k+1}^{n} u(d_i)$.

Example 53 is an example of non-local rewriting.

Example 53 (Non-local rewriting) *Let* μ_{NL} *be the unifier of* $q_{NL} = t(x_1,x_2) \wedge (s(x_1,x_3) \vee s(x_1,x_3)) \wedge (s(x_2,x_3) \vee s_1(x_2,x_3))$ *with* $R_2 = f(x) \rightarrow s(x,y)$ *defined by:* $\mu_{NL} = ((s(x_1,x_3) \vee s_1(x_1,x_3)) \wedge (s(x_2,x_3) \vee s_1(x_2,x_3)), s(x_1,x_3) \wedge s(x_2,x_3), u_{NL}(x_1) = u_{NL}(x_2) = x, u_{NL}(x_3) = y$. μ_{NL} *is not a local unifier, since it unifies two disjunctions at once. The rewriting of* q_{NL} *with respect to* μ_{NL} *is:*

$$f(x) \wedge t(x,x).$$

4.2.2 Gathering tools: COMPACT

Both local and non-local rewriting steps can be performed by using any kind of existential rules. However, to compute a first-order rewriting, this rewriting has to exist. We thus restrict the input of the algorithm we present in this section to arbitrary conjunctive

queries and *fus* rule sets [1]. COMPACT is thus an algorithm that takes as input a conjunctive query q and a *fus* $\mathcal{R}$ and outputs a sound and complete USCQ-rewriting of q with respect to $\mathcal{R}$. We first present a sub-procedure of COMPACT, which, given an SCQ s and a *fus* $\mathcal{R}$, saturates s with respect to local rewritings using rules of $\mathcal{R}$. We then present COMPACT, and apply it on the running example.

LU-SATURATION

LU-SATURATION is a sub-procedure of COMPACT that takes a set of rules and an SCQ as input, and outputs an SCQ on which no new local rewriting can be performed. Dealing with local rewritings in a special manner is not necessary for the soundness and completeness of Algorithm 4, but it is conceptually easier and is the first step towards an important optimization presented in the next section. Algorithm 3 presents this procedure: it only applies local rewritings as long as it is possible.

Algorithm 3: LU-SATURATION

Data: An SCQ s, a set of existential rules $\mathcal{R}$
Result: s saturated with respect to local unifications
$s_o := null$;
$s_n := s$;
while $s_o \neq s_n$ **do**
  $s_o := s_n$;
  for *every rule* $R \in \mathcal{R}$ **do**
    for *every local unifier* μ *of* R *with* s_n **do**
      $s_n := \gamma_L(s_n, R, \mu)$;
return s_n

Algorithm 3 terminates because of the $sep(d_1)$-entailment condition in Definition 4.10. Indeed, one can add at most pw^w atoms to a disjunction d_1 before any further atom to be $sep(d_1)$-entailed by d_1, where p is the number of predicates and w the maximum arity of a predicate. For the sake of clarity, we apply step by step Algorithm 3 on the running example.

Example 54 (Step by step application) *We apply Algorithm 3 on* $q_e = t(x_1,x_2) \wedge s(x_1,x_3) \wedge s(x_2,x_3)$ *and* $\mathcal{R}_e$. $R_1 = p(x) \wedge h(x) \rightarrow s(x,y)$ *has two atoms in its body, and thus no local unification is possible. Any unification of* q_e *with* $R_2 = f(x) \rightarrow s(x,y)$ *maps* x_3 *to an existentially quantified variable, thus should unify two atoms at once, and thus no local unification of* q_e *with* R_2 *exists.* $R_3 = f_1(x) \rightarrow s_1(x,y)$ *does not generate atoms of relevant predicate. There is a local unifier of* $R_4 = t(x,y) \rightarrow t(y,x)$ *with* q_e*, which unifies* $t(x_1,x_2)$*. Applying this unification modifies* s_n *to:*

$$(t(x_1,x_2) \vee t(x_2,x_1)) \wedge s(x_1,x_3) \wedge s(x_2,x_3).$$

1. Actually, COMPACT outputs a sound and complete rewriting as soon as it exists, even if $\mathcal{R}$ is not *fus*.

Two further local unifications of s_n *with* $R_5 = s_1(x,y) \rightarrow s(x,y)$ *exist (note that* x *and* y *are both frontier variables): one that unifies* $s(x_1,x_3)$ *and one that unifies* $s(x_2,x_3)$*. Applying the first unification results in setting* s_n *to:*

$$(t(x_1,x_2) \vee t(x_2,x_1)) \wedge (s(x_1,x_3) \vee s_1(x_1,x_3)) \wedge s(x_2,x_3),$$

and applying the second one updates s_n *to:*

$$(t(x_1,x_2) \vee t(x_2,x_1)) \wedge (s(x_1,x_3) \vee s_1(x_1,x_3)) \wedge (s(x_2,x_3) \vee s_1(x_2,x_3)).$$

s_n *having evolved, Algorithm 3 checks if any new unification is possible, which is not the case, for similar reasons as those explained above. The output of Algorithm 3 on* q_e *and* $\mathcal{R}_e$ *is thus:*

$$(t(x_1,x_2) \vee t(x_2,x_1)) \wedge (s(x_1,x_3) \vee s_1(x_1,x_3)) \wedge (s(x_2,x_3) \vee s_1(x_2,x_3)).$$

COMPACT

We now present Algorithm 4. Starting from an initial conjunctive query s, it performs a breadth-first exploration of the SCQ-rewritings of s. For each of these rewritings, it first performs an LU-saturation, and then generates new queries by rewriting using non-local unifiers. We could consider only non-local *prime* unifiers, as defined and explained in the next subsection. A cover of the set of SCQ-rewritings is computed. The priority is given to already explored queries: during the computation of the cover, if two queries s_1 and s_2 are equivalent, and s_1 has been explored (necessarily s_2 has not been explored yet), we keep s_1 in the cover. Remaining non-explored queries are explored.

We illustrate COMPACT by presenting a step by step application of this algorithm on the running example.

Example 55 (Step by step application of Algorithm 4) *We first LU-saturate* q_e*, modifying* q_e *to:*

$$q_e = (t(x_1,x_2) \vee t(x_2,x_1)) \wedge (s(x_1,x_3) \vee s_1(x_1,x_3)) \wedge (s(x_2,x_3) \vee s_1(x_2,x_3)),$$

as explained in Example 54. We now create new SCQs, resulting from non-local unifications of q_e *with some rule of* $\mathcal{R}_e$*.* q_e *is unifiable with* $R_1 = h(x) \wedge q(x) \rightarrow s(x,y)$*, by considering the selection* $s(x_1,x_3) \wedge s(x_2,x_3)$*. The resulting SCQ is* $q_e^1 = h(x) \wedge q(x) \wedge t(x,x)$*. By considering the same selection,* q_e *is unifiable with* $R_2 = p(x) \rightarrow s(x,y)$*, which results in* $q_e^2 = p(x) \wedge t(x,x)$*.* q_e *is unifiable with* $R_3 = p_1(x) \rightarrow s_1(x,y)$ *(with corresponding selection* $s_1(x_1,x_3) \wedge s_1(x_2,x_3)$*), resulting in* $q_e^3 = p_1(x) \wedge t(x,x)$*. Last,* q_e *is unifiable with* R_5 *with selection* $s(x_1,x_3) \wedge s(x_2,x_3)$*, resulting in* $q_e^4 = s_1(x,y) \wedge t(x,x)$*.*

Algorithm 4 does not generate any further query at this step. $q_e^4 \leq q_e$*, and is thus discarded. All other SCQs are pairwise incomparable, and no further unification is possible: Algorithm 4 stops and outputs* $\{q_e, q_e^1, q_e^2, q_e^3\}$*.*

Algorithm 4: COMPACT

Data: A CQ s (thus a SCQ), a *fus* $\mathcal{R}$
Result: A sound and complete USCQ-rewriting of s w.r.t. $\mathcal{R}$
$\mathcal{S}_F := \{s\}$; \\ the set of final queries
$\mathcal{S}_E := \{s\}$; \\ the set of queries to be explored
while $\mathcal{S}_E \neq \emptyset$ **do**
  $\mathcal{S}_t := \emptyset$; \\ the set of queries generated during the current step
  for *every* $s' \in \mathcal{S}_E$ **do**
    $s' :=$ LU-SATURATION(s');
    for *every rule* $R \in \mathcal{R}$ **do**
      for *every non-local (prime) unifier* μ *of* R *with* s' **do**
        $\mathcal{S}_t := \mathcal{S}_t \cup \{\gamma_{NL}(s', R, \mu)\}$
  $\mathcal{S}_t :=$ cover$(\mathcal{S}_F \cup \mathcal{S}_t)$;
  $\mathcal{S}_E := \mathcal{S}_t \setminus \mathcal{S}_F$;
  $\mathcal{S}_F := \mathcal{S}_t$;
return $\mathcal{S}_F$

Theorem 8 *Algorithm 4 computes a sound and complete rewriting for any query* q *and any* fus.

The proof of Theorem 8 uses following Lemmas 2 and 3. We first give these lemmas and corresponding proofs, then present the proof of Theorem 8. Lemma 2 is a tool to prove that the output of COMPACT is sound. It states that any selection of a rewriting of an SCQ s is (less general than) a rewriting (with the usual piece-based rewriting operator) of a selection of s. It will thus allow to use the soundness of usual piece-based rewriting.

Lemma 2 *Let* s *be an SCQ,* R *be an existential rule, and* μ *be a unifier of* s *with* R*. Then for any selection* q' *of a rewriting* $\gamma_L(s, R, \mu)$ *(or* $\gamma_{NL}(s, R, \mu)$ *if* μ *is not local), there exists a selection* q *of* s *such that* q'*is a rewriting of* q.

Proof: Let $\mu = (s_\mu, q_\mu, u)$ be a unifier of s with R, and q' be a selection of $\beta(s, R, \mu)$. $q' = q'_1 \wedge q'_2$, where q'_1 is a selection of $u(s \setminus s_\mu)$, and q'_2 comes from novel disjunctions (or disjunctions to which an atom has been added if μ is local). Let q be defined as the conjunction of the inverse image of q'_1 by u, denoted by q_1 and q_μ. q is a selection of s (q_1 can be selected in $s \setminus s_\mu$, and q_μ in s_μ). Moreover, $\mu_{cq} = (q_\mu, u)$ is a unifier of q with R, since q_μ is non-empty, the piece conditions are fulfilled by assumption on μ being a unifier, as well as the unifying assumption (same atoms, same substitution). The rewriting of q with respect to μ_{cq} is equal (up to variable renaming) to q'. □

Lemma 3 is a tool to prove that the output of COMPACT is complete.

Lemma 3 *Let* s *be an SCQ,* q *be a selection of* s*, and* R *be a rule. For any rewriting* q' *of* q *with* R*, there exists a rewriting* s' *of* s *with* R *and a selection* q'' *of* s' *such that* q' *is equivalent to* q''.

Proof: q' is a rewriting of q, thus there exists a rule R and unifier $\mu = (q_u, u)$ of q with R such that $q' = \beta(q, R, \mu)$. Let k be the number of atoms of q_u. q is a selection of s, and without loss of generality, we can set $s = \bigwedge_{i=1}^{n} d_i$, with q_u being a selection of $s' = \bigwedge_{i=1}^{k} d_i$. Let $\mu_{SCQ} = (s', q_u, u)$. μ_{SCQ} is a unifier of s with R. Indeed:

- s' is a non-empty subset of s
- q_u is a selection of s' (by construction of s')
- u being unchanged, Conditions 1 and 3 of Definition 4.7 are fulfilled
- the separator of s' w.r.t s is equal to the separator of q_u with respect to q, thus Condition 2 is also fulfilled.

If μ_{SCQ} is local, it implies that q is equal to q' except for one atom. This atom is the image of the body of the rule, and thus q' is a selection of $\gamma_L(s, R, \mu_{SCQ})$ by selecting the newly added atom (the instantiated body of the rule) and in other disjunctions, the same atoms as q. If μ_{SCQ} is not local, essentially the same selection applies: we select every newly created atom, and the image of the atoms of q in the disjunction for which no atom have been unified. The obtained query is homomorphically equivalent to q'. □

We can now prove the correctness of Algorithm 4.

Proof of Theorem 8: the soundness of Algorithm 4 relies on the fact that every SCQ that is generated by Algorithm 4 is a sound rewriting of the input s. To show this, we notice that an SCQ s' is a sound rewriting of a conjunctive query q if and only if every selection of s' is a sound rewriting of q. q is a sound rewriting of q, and Lemma 2 shows that a piece-based rewriting of a sound rewriting of q is a sound rewriting of q.

As for the completeness of Algorithm 4, we prove that after going i times through the while loop, any $\mathcal{R}$-rewriting obtained after less than i rewriting step (of *depth* less than i) is less general than an element of $\mathcal{S}_F \cup \mathcal{S}_E$, *i.e.*, covered by $\mathcal{S}_F \cup \mathcal{S}_E$. Initially, one should check that the only $\mathcal{R}$-rewriting of depth 0 of s, which is s, is covered by $\{s\}$. Let us assume that after going i times through the while loop, all $\mathcal{R}$-rewritings of depth i are covered by $\mathcal{S}_F \cup \mathcal{S}_E$. Let q_{i+1} be an $\mathcal{R}$-rewriting of s of depth $i+1$. There exists q_i of depth i, R_i and μ_i a unifier of q_i with R_i such that $q_{i+1} = \beta(q_i, R_i, \mu_i)$. By induction assumption, q_i is covered by $\mathcal{S}_F^i \cup \mathcal{S}_E^i$, and thus there are $s_i \in \mathcal{S}_F^i \cup \mathcal{S}_E^i$ and q_i^* a selection of s_i such that $q_i^* \geq q_i$. If $q_i^* \geq q_{i+1}$, we define $q_{i+1}^* = q_i^*$. Otherwise, there exists a rewriting q_{i+1}^* of q_i^* such that $q_{i+1}^* \geq q_{i+1}$. By Lemma 3, there exists a rewriting of s_i, that we denote by s_{i+1}, such that s_{i+1} contains a selection that is more general than q_{i+1}^*. Since any rewriting of $\mathcal{S}_F^i \cup \mathcal{S}_E^i$ is covered by $\mathcal{S}_F^{i+1} \cup \mathcal{S}_E^{i+1}$, the output of Algorithm 4 is complete. □

We already discussed that the number of SCQs in a USCQ may not be a good parameter for evaluating the quality of a USCQ-rewriting. It is still worth noticing that COMPACT outputs USCQs which are not necessarily optimal with respect to this size notion, as can be seen with the running example.

Example 56 *Let $\mathcal{S}$ be a set of SCQs containing exactly the following queries:*

- $(t(x_1, x_2) \vee t(x_2, x_1)) \wedge (s(x_1, x_3) \vee s_1(x_1, x_3)) \wedge (s(x_2, x_3) \vee s_1(x_2, x_3))$,
- $t(x, x) \wedge h(x) \wedge q(x)$,

– $t(x,x) \wedge (p(x) \vee p_1(x))$.

Let us recall that the output of COMPACT *in Example 55 contains the following queries:*

– $q_e = (t(x_1,x_2) \vee t(x_2,x_1)) \wedge (s(x_1,x_3) \vee s_1(x_1,x_3)) \wedge (s(x_2,x_3) \vee s_1(x_2,x_3))$
– $q_e^1 = t(x,x) \wedge h(x) \wedge q(x)$
– $q_e^2 = t(x,x) \wedge p(x)$
– $q_e^3 = t(x,x) \wedge p_1(x)$

$\mathcal{S}$ *is of strictly smaller cardinality than the output of* COMPACT*, but is an equivalent formula. Indeed, the first two queries of* $\mathcal{S}$ *belongs to the output of* COMPACT*, and the third one is equivalent to the disjunction of* q_e^2 *and* q_e^3*. This proves that* COMPACT *is not optimal in terms of the number of SCQs in the output USCQ.*

Can we design an optimal algorithm for computing USCQ-rewritings? This is a natural question, but we leave if for further work, due to the following observations:

– one of the strengths of COMPACT is its simplicity - the only lead found so far to aim at optimality makes it significantly more complex;
– meanwhile, the number of SCQs output by COMPACT on benchmarks is small - see the evaluation section;
– small size is not a guarantee of efficient evaluation;
– characterizing the optimal sound and complete USCQ-rewriting is not as easy as in the UCQ case.

Indeed, we showed in Theorem 7 that to check whether a sound and complete UCQ-rewriting is minimal, it is sufficient to check if two elements are comparable for $\geq$. This property is not true anymore with USCQ-rewritings: Example 56 presents two sound and complete USCQ-rewritings that both fulfill the condition of not having two comparable elements, while not being of the same cardinality.

An optimization: prime unifiers

The breadth-first backward chaining presented in the previous section is sound and complete, but part of the performed rewritings are useless. This, of course, is undesirable, and we suggest a simple optimization that removes some of these useless computations. This optimization is based on the following observation.

Example 57 (A useless rewriting) *In Example 55,* $q_e^4 = t(x,x) \wedge s_1(x,y)$ *is obtained from* $q_e = t(x_1,x_2) \wedge s(x_1,x_3) \wedge s(x_2,x_3)$ *by a rewriting with respect to* $R_5 = s_1(x,y) \rightarrow s(x,y)$ *and* $\mu_{NL} = (s(x_1,x_3) \wedge s(x_2,x_3), s(x_1,x_3) \wedge s(x_2,x_3), u_{NL}(x_1) = u_{NL}(x_2) = x, u_{NL}(x_3) = y)$*. Let us now "split"* μ_{NL} *into two local unifiers* μ_L^1 *and* μ_L^2 *defined as follows:*

– $\mu_L^1 = (s(x_1,x_3), s(x_1,x_3), u_L^1(x_1) = x, u_L^1(x_3) = y)$
– $\mu_L^2 = (s(x_2,x_3), s(x_2,x_3), u_L^2(x_2) = x, u_L^2(x_3) = y)$

The rewriting with respect to μ_{NL} *is less general than the rewriting* $t(x_1,x_2) \wedge (s(x_1,x_3) \vee s_1(x_1,x_3)) \wedge (s(x_2,x_3) \vee s_1(x_2,x_3))$ *obtained after rewriting with respect to* μ_L^1 *then* μ_L^2*.*

This observation motivates the definition of *prime unifier*, which is intuitively a unifier for which no part could be locally done.

Definition 4.12 (Prime unifier) *Let* s *be an SCQ,* R *be a rule and* $\mu = (s', q', u)$ *be a non-local unifier of* s *with* R. μ *is* prime *if for any* $d \in s'$*, and for any* u_L*, substitution of* $fr(R) \cup vars(d)$ *by terms*$(head(R)) \cup \mathcal{C}$*,* $\mu = (\{d\}, q_d, u_L)$*, where* q_d *is the atom of* q' *that has been selected from* d*, is not a local unifier of* s *with* R.

Example 58 μ_{NL} *in Example 57 is an example of non-prime unifier. A prime unifier that unifies more than one disjunction is the following unifier* μ_p *of* $q_e = t(x_1, x_2) \wedge s(x_1, x_3) \wedge s(x_2, x_3)$ *with* $R_2 = f(x) \rightarrow s(x, y)$*:* $\mu_p = (s(x_1, x_3) \wedge s(x_2, x_3), s(x_1, x_3) \wedge s(x_2, x_3), (u(x_1) = u(x_2) = x, u(x_3) = y))$. *Indeed, none of both unified atoms could be locally unified, since* x_3 *should be unified with* y*, which is an existentially quantified variable - and thus both atoms should be unified together.*

Following Property 35 ensures that we can consider only prime unifiers – since any query obtained by using a non-prime unifier is less general than a query obtained using local rewritings and a single prime rewriting.

Property 35 *Let* $\mu = (s' \cup \{d\}, q' \cup \{a\}, u)$ *be a unifier of* s *with* R *such that* $\mu_L = (d, a,$ *mgu*$(a,$ *head*$(R)))$ *is a local unifier of* s *with* R*. Let* u' *be the restriction of* u *to terms*$(q') \cup fr(R)$*. Then* $\mu' = (s', q', u')$ *is a unifier of* $s_L = \beta(s, R, \mu_L)$ *with* R *and:*

$$\beta(s_L, R, \mu') \geq \beta(s, R, \mu).$$

Proof: Let us first prove that μ' is a unifier of s_L with R. We check the three conditions expressed in the definition of piece-unifier. Since $u(s' \cup \{d\}) \subseteq u(\text{ head}(R))$, it also holds for $u(s')$. An element x of $\text{sep}(s')$ in s_L was either a separator of $s' \cup \{d\}$ in s, or it is a variable shared by s' and d. In the first case, since μ is a unifier, it implies that $u(x) \in \text{fr}(R) \cup \mathcal{C}$. In the second case, since μ_L is a unifier, it means that x is unified with an element of $\text{fr}(R) \cup \mathcal{C}$. In both cases, the second condition is also fulfilled. u' is equal to u on $\text{fr}(R)$, which shows that the first condition is fulfilled.

We have shown that μ' is a unifier of s_L with R. We now prove the second part of the claim: $\beta(s_L, R, \mu') \geq \beta(s, R, \mu)$. We first express both rewritings. We have $\beta(s_L, R, \mu') = u'(\text{ head}(R)) \wedge u'(d \vee (h')) \wedge u'(s \setminus (s' \cup \{d\}))$. On the other hand, $\beta(s, R, \mu) = u(\text{ head}(R)) \wedge u(s \setminus (s' \cup \{d\}))$. We first notice that for any selection of $\beta(s_L, R, \mu')$, there is a selection of $u'(d \vee \{h'\}) \wedge u'(s \setminus (s' \cup \{d\}))$ which is more general – indeed, the atom $u'(\text{ head}(R)$ can be mapped on h' by using a homomorphism equal to the identity on shared terms, we extend this substitution by the identity on any other term. Last, u' being more general than u, $u(\text{ head}(R)) \wedge u(s \setminus (s' \cup \{d\})$ is less general than $u(\text{ head}(R)) \wedge u(s \setminus (s' \cup \{d\}))$, which concludes the proof.

□

Property 36 *Algorithm 4 outputs a sound and complete rewriting of* q *with* $\mathcal{R}$ *even if only rewritings with respect to prime unifiers are computed.*

Proof: Property 35 ensures that for any SCQ s and any one step-rewriting s' of s that is obtained thanks to a non-prime unifier, there exists s'' with $s'' \geq s'$ that is obtained by a step of LU-saturation of s and a single rewriting with a prime-unifier. The proof of Theorem 8 can thus be directly adapted to the case where only prime-unifiers are used. □

Let us last present a property showing that COMPACT behaves well with respect to large relation hierarchies: when the ontology is restricted to such axioms, COMPACT is guaranteed to generate a USCQ restricted to a single SCQ, thanks to local unifications.

Property 37 *Let $\mathcal{R}$ be a set of rules of the following shape:*

$$p(x_1,\ldots,x_k) \rightarrow q(y_1,\ldots,y_n),$$

such that x_i are distinct variables, y_i are distinct variables, $\{y_i\} \subseteq \{x_i\}$. Let q be a conjunctive query. The sound and complete USCQ-rewriting of q w.r.t. $\mathcal{R}$ is a single SCQ.

Proof: Let s be an SCQ. We show that for any $R \in \mathcal{R}$, there is no prime unifier of R with s. This is sufficient to show the claim, since only prime unifiers can generate new SCQs. First, let $\mu = (s', q', u)$ such that s' is of cardinality 1, *i.e.*, contains only one disjunction. u is a most general unifier of $\text{head}(R)$ with q', which is unique up to variable renaming. Since no variable of $\text{head}(R)$ appears twice, u is injective on the variables of q'. If μ is such that s' contains at least two disjunctions, then for any $d_i \in s'$, $\mu_i = (\{d_i\}, q(d_i), u_{|\ \text{vars}(q(d_i))})$ is a unifier of R with S, since R is range-restricted[2]. We conclude as above that μ is not prime. □

4.2.3 A close-up on the subsumption test

A difficulty is hidden in Algorithm 4: an efficient computation of a cover of a set of SCQs requires an efficient computation of the subsumption test. However, such a computation is more complex than it is for conjunctive queries, where a single homomorphism check is sufficient to compare two queries.

Property 38 (Subsumption for SCQs) *Let s_1 and s_2 be two SCQs. $s_1 \geq s_2$ if for any selection q_2 of s_2, there exists a selection q_1 of s_1 such that $q_1 \geq q_2$.*

Thus, the brute-force way to check whether $s_1 \geq s_2$ would be to compute all selections of s_2, and to check for each of them whether there exists a selection of s_1 which is more general. This is not acceptable, since our aim is precisely to avoid considering every possible selection, keeping a compact representation of unions of conjunctive queries.

2. We recall that range-restricted rules do not contain any existential variable in the head.

We first take a closer look at the structure of the SCQs that are generated by COMPACT. The key notion is the following: in any SCQ generated during the rewriting process, the disjunctions are very specific, since for any of them, there exists an atom that can "generate" all other atoms by being rewritten using local unifiers. If moreover it can generated using only range-restricted rules, we call such an atom an $\mathcal{R}$-root, and it will be used to prune the exploration space during the subsumption test. Let us present the formal definition of a root.

Definition 4.13 ($\mathcal{R}$-root) *Let $\mathcal{R}$ be a set of rules, $s = \bigwedge_i d_i$ be an SCQ. Let X_i be the separator of d_i in s. Let a be an atom of d_i, and $s_a = \bigwedge_{j \neq i} d_j \wedge a$. Let $s'_a = \bigwedge_j d'_j$ be the* LU-SATURATION *of s_a by range-restricted rules. d_i admits atom a as a* root*, if for any atom b of d_i, b is X_i-entailed by d'_i. s is* fully rooted *if every disjunction of s is rooted.*

Example 59 illustrates the notion of $\mathcal{R}$-root on the running example.

Example 59 ($\mathcal{R}$-root) *Let q^s_e be the local saturation of q_e:*

$$q^s_e = (t(x_1,x_2) \vee t(x_2,x_1)) \wedge (s(x_1,x_3) \vee s_1(x_1,x_3)) \wedge (s(x_2,x_3) \vee s_1(x_2,x_3)).$$

$\{t(x_1,x_2), t(x_2,x_1)\}$ admits two $\mathcal{R}_e$-roots, which are $t(x_1,x_2)$ and $t(x_2,x_1)$. Indeed, $t(x_1,x_2)$ can be obtained by locally rewriting $t(x_2,x_1)$ thanks to the rule $t(x,y) \rightarrow t(y,x)$, and conversely. $\{s(x_1,x_3), s_1(x_1,x_3)\}$ admits a single $\mathcal{R}_e$-root, which is $s(x_1,x_3)$, since $s_1(x_1,x_2)$ can be obtained by a local rewriting using the rule $s_1(x,y) \rightarrow s(x,y)$. The converse is not possible. The situation is similar for the third disjunction.

The goal is to consider only $\mathcal{R}$-roots when checking subsumption. This, however, is not directly possible. Indeed, let us consider the rule $R : p(x) \rightarrow r(x,x)$, and the two SCQs $s_1 = p(x) \vee r(x,x)$ and $s_2 = r(x,y)$. The only disjunction of s_1 admits an $\mathcal{R}$-root, which is $r(x,x)$, and s_2 also admits an $\mathcal{R}$-root, which is $r(x,y)$. The $\mathcal{R}$-root of s_2 is more general than the $\mathcal{R}$-root of s_1, but the LU-Saturation of s_2 (which is equal to s_2) is not more general than s_1. This is due to the fact that the considered homomorphism is not injective, and unification that were not local on s_2 may become local on s_1. An on-the-fly processing of such a case is possible – however, we propose here only a very simple solution, which happened to be sufficient on the benchmarks.

Indeed, we only consider *simple* roots, that are roots for which the behavior we exhibited above cannot appear, because there are neither duplicated terms nor constants from the rule set.

Definition 4.14 (Simple root) *A root is* $\mathcal{R}$-simple *if it does not have any constant appearing in a rule of $\mathcal{R}$ as argument, and if no term appears twice among its arguments.*

Given an SCQ, we are interested in its $\mathcal{R}$-simplification: for each disjunction that admits a simple root, we replace the whole disjunction by its simple root.

Definition 4.15 ($\mathcal{R}$-simplification of an SCQ) *Let $\mathcal{R}$ be a set of rules, $s = \bigwedge_i d_i$ be an SCQ. An $\mathcal{R}$-simplification of s is an SCQ $s' = \bigwedge_i d'_i$ such that for any i, $d'_i = \{a\}$, where a is one of the simple roots of d_i if such a root exists, and $d'_i = d_i$ otherwise.*

Note that the $\mathcal{R}$-simplification of an SCQ is less general than the SCQ itself. Example 60 presents an $\mathcal{R}$-simplification in the framework of our running example.

Example 60 ($\mathcal{R}$-simplification) *q^s_e in Example 59 admits two $\mathcal{R}$-simplifications, which are:*

$$t(x_1,x_2) \wedge s(x_1,x_3) \wedge s(x_2,x_3),$$

and

$$t(x_2,x_1) \wedge s(x_1,x_3) \wedge s(x_2,x_3).$$

Property 39 *Let s_1 be an LU-saturated SCQ and s_2 be an SCQ. $s_1 \geq s_2$ iff there exists an $\mathcal{R}$-simplification s'_2 of s_2 such that $s_1 \geq s'_2$.*

Proof: Let $s_2 = \bigwedge_i d_i$ be an SCQ, such that there are a and b in d_1 such that b can be obtained by local unification from a, and a has no argument appearing twice and no constant appearing in $\mathcal{R}$ as argument. We show that $s_1 \geq s'_2$ if and only if $s_1 \geq s_2$, where $s'_1 = d'_1 \wedge \bigwedge d_i$, where $d'_1 = d_1 \setminus \{b\}$, which will imply the property by induction on the number of atoms that are removed to obtain the $\mathcal{R}$-simplification of s_2.

Since $s_2 \geq s'_2$, $s_1 \geq s_2$ implies $s_1 \geq s'_2$. We thus focus on the converse, and assume that $s_1 \geq s'_2$. Let q_2 be a selection of s_2. Either q is also a selection of s'_2, and thus there exists a selection q_1 of s_1 such that $q_1 \geq q_2$. Or, there exists c such that:

$$q_2 = b \wedge \bigwedge_{i>1} c(d_i).$$

Let q'_2 be the following conjunctive query:

$$q'_2 = a \wedge \bigwedge_{i>1} c(d_i).$$

q'_2 is a selection of s'_2, and as such, there exists a selection q'_1 of s_1 such that $q'_1 \geq q'_2$, which implies that there is a homomorphism π_1 from q'_1 to q'_2. If $\pi_1(q'_1) \subseteq \bigwedge_{i>1} c(d_i)$, then π_1 is also a homomorphism from q'_1 to q_2. Otherwise, there exists an atom a' such that $\pi_1(a') = a$. Since a has no constant of $\mathcal{R}$ as argument and no variable appearing twice, so is it for a'. Moreover, b is obtained from a thanks to a range-restricted rules, so b' more general than b can be obtained from a' with a local unification. Thus, $q_1 = b \wedge \bigwedge_{i>1} c(d_i)$ is a selection of s_1 such that $q_1 \geq q_2$, which shows the claim. □

This simple optimization has been implemented in COMPACT. During the computation of the cover, an $\mathcal{R}$-simplification is computed for each query. We then use Property 39 to check the subsumption relations, by exploding $\mathcal{R}$-simplifications and comparing obtained conjunctive queries to other SCQs.

4.3 Experimental evaluation of COMPACT

We now provide first experiments to evaluate COMPACT. As already pointed out, it is important to evaluate both the rewriting step and the querying step. Indeed, the considered approach would be much less interesting if, by making the rewriting phase quicker, one would significantly make the querying phase longer. However, making this evaluation represents a challenge, for the following reasons.

Idiosyncrasies of COMPACT. COMPACT stands out in the landscape of rewriting tools, for two main reasons. First, this algorithm and the piece-based rewriting algorithm of [König *et al.*, 2012] are the only two algorithms that can generate sound and complete rewritings for *any fus*, and not only for specific subclasses. Indeed, most rewriting tools are able to create rewritings only for specific subclasses, typically linear or sticky rules. Second, it is the only tool that generates a union of semi-conjunctive queries, and not a union of conjunctive queries or a Datalog program. Summing up, we have two problems:

- the comparison with state-of-the-art tools would not take into account the greater generality of COMPACT,
- a structural comparison of the outputs of COMPACT and of other algorithms is not straightforward.

Missing benchmarks. A second major problem is the lack of appropiate benchmarks. The benchmark classically used since [Pérez-Urbina *et al.*, 2009] to evaluate query rewriting algorithms is composed of five ontologies, with five queries each. The most used ontology is LUBM[3], and several adaptations to it have been proposed [Rodriguez-Muro and Calvanese, 2012; Lutz *et al.*, 2012]. The small number of queries in the benchmark is already a serious weakness. This has already been noticed, and a very recent paper proposed an automatic generation of relevant queries in order to test soundness and completeness of algorithms [Imprialou *et al.*, 2012]. However, even the ontologies in themselves are questionable. Most of the rules simply translate class or role hierarchies, and only a few rules contain existentially quantified variables. Last, they are all atomic-body rules, which makes these ontologies belong to a small part of those covered by COMPACT.

We believe that establishing a proper and *meaningful* benchmark would be a very interesting contribution to the field. However, this is a not an easy task: randomly generated ontologies would not qualify as meaningful, and finding parameters to guide the randomness is non-trivial. Getting to applications to build real-world ontologies is definitely outside of the scope of this thesis. Even aware of these weaknesses, we did not have other choices than to stick to the evaluation protocol of query rewriting algorithms currently accepted by the community.

3. http://swat.cse.lehigh.edu/projects/lubm/

In the remaining of that section, we first evaluate the rewriting step of COMPACT, by comparing the output of COMPACT with the output of Iqaros [Venetis *et al.*, 2012]. Then, since COMPACT outputs a USCQ-rewriting and not a UCQ-rewriting, we compare the evaluation of both outputs. The data consists of LUBM-generated data, with 20 universities (for a total of 556k unary atoms and 2,2M binary atoms). All tests have been performed on a 2.4GHz processor, with 4GB of RAM. The RDMS used is Sqlite.

4.3.1 Rewriting step evaluation

We considered the LUBM ontology to evaluate the rewriting step of COMPACT. However, this ontology being rather flat, we created a family of variants, LUBM_n, obtained as follows. We added, for each class (resp. each role) of the LUBM ontology n subclasses (resp. n subroles). As for queries, in order to have a wider evaluation basis than the five queries of the original protocol [Pérez-Urbina *et al.*, 2009], we also used Sygenia,[4] the query generator described in [Imprialou *et al.*, 2012], to generate new queries.

On the benchmark that we consider, Iqaros is the most performant existing tool. We thus compare COMPACT with Iqaros. Since the outputs of COMPACT and Iqaros are of different kinds, we are mainly interested in the time required to compute a sound and complete U(S)CQ-rewriting. The comparison of the quality of the outputs, usually done by counting the number of conjunctive queries in a UCQ-rewriting, will be done in the next section, by assessing how efficiently each rewriting can be evaluated.

4.3.2 Querying step evaluation

We now focus on the evaluation of the querying step. To that purpose, we evaluate both the USCQ generated by COMPACT and the equivalent UCQ.

Assumptions and experimental settings

Given that most of the data are available in relational databases, we stored the data in a relational database. We assumed that for each predicate of arity k, there is a corresponding table of arity k whose columns are named $c1, \ldots, ck$. While this is a simplifying assumption, it does not change the fundamental problem of query rewriting, which is the one we focus on. The ontologies and queries we consider in this section are the same as in the previous section, and the presentation of results is once again split between handcrafted queries and automatically generated queries - which are of small size. The data we use is the data generated by the LUBM data generator[5] – this is a weakness of our evaluation, since the concepts introduced in LUBM_n have no associated data.

4. Available at http://code.google.com/p/sygenia/
5. Also available at http://swat.cse.lehigh.edu/projects/lubm/.

```
CREATE VIEW disj1 AS
SELECT c1,c2 AS c1,c2 FROM t
UNION
SELECT c2,c1 AS c1,c2 FROM t
```

Figure 4.1: The view associated with the SCQ: $(t(x_1,x_2) \vee t(x_2,x_1))$

SQL translations

We present here how we translate semi-conjunctive queries into SQL queries. It significantly differs from the standard translation for conjunctive queries by the use of views to compute unions of tables. Indeed, semi-conjunctive queries are translated by a two-step process. We first replace each disjunction by a view. The view is basically equal to the disjunction of every relation that appear in the disjunction – care should be taken about the orders of the arguments. Then, the SCQ can be seen as a conjunctive query on the names of the tables, and the previous translation is applied. We illustrate this two-step process on the LU-saturation of q_e:

$$(t(x_1,x_2) \vee t(x_2,x_1)) \wedge (s(x_1,x_3) \vee s_1(x_1,x_3)) \wedge (s(x_2,x_3) \vee s_1(x_2,x_3)).$$

We create three views, named $\mathtt{disj1}$, $\mathtt{disj2}$ and $\mathtt{disj3}$ representing each disjunction. Figure 4.1 presents the SQL query that is associated with $(t(x_1,x_2) \vee t(x_2,x_1))$. We then evaluate the query $\mathtt{disj1}(x_1,x_2) \wedge \mathtt{disj2}(x_1,x_3) \wedge \mathtt{disj3}(x_2,x_3)$, which is classically translated into a SQL query. Union of conjunctive queries are evaluated in a naive way: each conjunctive query of the union is evaluated separately.

4.3.3 Experimental results

Results. Experimental results are presented in Tables 4.1 and 4.2 as well as in Figures 4.2 and 4.3. Results are aggregated by ontologies – a single line (or column in Figure 4.2 and 4.3) represent all queries relative to an ontology. Tables 4.1 and 4.2 present the time needed by COMPACT and Iqaros to rewrite respectively Sygenia-generated queries and queries that come from the original benchmark. The ontologies are LUBM_n, for n from 0 to 8. The number of selections is also mentioned. It is worth to note that, on this benchmark, the number of selections of the USCQ output by COMPACT is equal to the number of conjunctive queries output by Iqaros. There is no theoretical guarantee of this correspondence in the general case, and this may be due to the simplicity of the benchmark. However, generating the output as a USCQ is much more faster than generating a UCQ – up to a factor 500 on the benchmark. Figures 4.2 and 4.3 present the time, in seconds, needed to evaluate the optimal UCQ-rewriting (black bars), and the USCQ-rewriting (white bars), for Sygenia-generated queries and for handcrafted queries, respectively. Missing bars represent a timeout, that has been fixed to 30 minutes. This suggests that the performed reformulation has also been beneficial for the querying step.

Table 4.1: Rewriting time and output for Sygenia queries

	COMPACT			Iqaros	
	# SCQs	# Selections	Time (ms)	# CQ	Time (ms)
0	102	486	44	486	152
1	102	1203	56	1203	171
2	102	1910	75	1910	182
3	102	2691	68	2691	205
4	102	3546	86	3546	257
5	102	4475	108	4475	342
6	102	5478	144	5478	440
7	102	6555	173	6555	556
8	102	7706	217	7706	692

Table 4.2: Rewriting time and output for handcrafted queries

	COMPACT			Iqaros	
	# SCQs	# Selections	Time (ms)	# CQ	Time (ms)
0	5	19	68	19	200
1	5	102	73	102	247
2	5	360	143	360	460
3	5	972	213	972	1454
4	5	2190	303	2190	5242
5	5	4338	406	4338	17382
6	5	7812	530	7812	56095
7	5	13080	667	13080	155566
8	5	20682	823	20682	403229

Limitations. Beyond the already discussed questionable representativity of the used benchmark, these results should be taken with care. Indeed views using disjunction are usually considered to behave poorly from an efficiency point of view. The obtained results are thus quite surprising. It should come from the fact that introduced tables have not been populated. Moreover, other experiments using a another RDMS (Sqlite is currently used) should be performed. Despite these limitations, we believe that USCQs are a good starting point for the evaluation of sound and complete rewritings.

4.4 Related and further work

We finish this chapter by presenting other tools that perform query rewriting. Since the seminal work on query rewriting with DL-Lite [Calvanese *et al.*, 2007], a number of rewriting tools have been implemented.

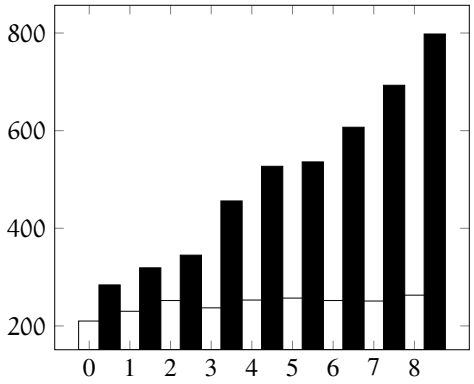

Figure 4.2: Querying time for Sygenia generated queries (in seconds)

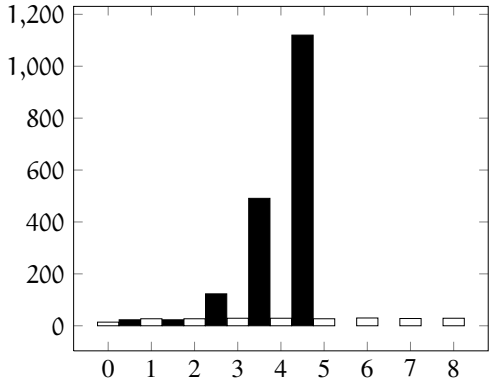

Figure 4.3: Querying time for handcrafted queries (in seconds)

The most common approach, on which we based our work, is to generate UCQ-rewritings. Among implemented tools, we can cite QuOnto [Acciarri *et al.*, 2005], Nyaya [Gottlob *et al.*, 2011], Rapid [Chortaras *et al.*, 2011], Iqaros [Venetis *et al.*, 2012], or the piece-based rewriting algorithm presented in [König *et al.*, 2012]. The rewriting operation used by Nyaya (which can process sticky rules in addition to linear rules) is very close from piece-based rewriting – however, this operation is split into two parts (applicability and factorization). Rapid takes as input only a subclass of linear rules. It first generates "structures" of rewritten queries, and then rewrites every atom of the query. These two steps are close from non-local and local rewritings, with the notable exception that Rapid generates all structures at once, which is possible thanks to the very special structure of ontologies that are dealt with by Rapid. Last, Iqaros deals with linear rules and performs an incremental rewriting, where atoms are added and rewritten one at at time.

The size of rewritings has also been studied in the literature, with a particular interest for cases where UCQ-rewritings are ensured to be of polynomial size. In particular, [Kikot *et al.*, 2011] exhibits a class of rules ensuring this property.

Another approach is to rewrite the query into Datalog programs. Most of the results here are of theoretical nature: the existence of a polynomial rewriting is proved for $\mathcal{EL}$ ontologies (for which no finite UCQ-rewriting is guaranteed to exist), and for sticky and linear rules [Gottlob and Schwentick, 2012]. A tool is however available to perform query rewriting into Datalog programs for DL-Lite ontologies: Presto [Rosati and Almatelli, 2010].

Last, let us mention that is has been shown recently [Imprialou *et al.*, 2012] that none of the available implementations were sound and complete (for rewriting into UCQs, in 2012). This intriguing result leads us to stress out two points:

- as already said, benchmarks are missing, and this hinders the development of sound and complete tools;
- the tests done in [Imprialou *et al.*, 2012] do not mean that *algorithms* were not sound and complete, but that their *implementation* was bugged. This strongly supports the idea of the need for simple algorithms, that do not require complex optimizations to run in a reasonable amount of time.

There is still a lot of work to be done. First, the performance of the implementation of COMPACT could be significantly enhanced, by performing parts of the computation offline and storing these results. Beyond such easy improvements, it would be very interesting to test the behavior of COMPACT on other ontologies and queries, as well as with more realistic data. Even for ontologies that do not belong to known first-order rewritable classes, COMPACT will output sound and complete rewritings as long as they exists. Does it happen often? Does COMPACT computes them efficiently as well? We conjecture that the biggest obstruction would be the computation of the cover operation: the optimization proposed in this dissertation allowed us to compute it reasonably efficiently (by reducing this to a "simple" homomorphism check). Further optimizations may be needed in a more general case. Moreover, a more in depth comparison with other systems, such as Iqaros, Rapid, Nyaya and Presto may bring some new ideas of optimizations.

Other questions naturally arise. What happens if we allow ourselves to use a more powerful querying language, such as query path language? Transitivity rules would not be an obstacle to rewritability anymore. A first step in that direction has already been undertaken in [Rudolph and Krötzsch, 2013], but no system has been implemented yet.

CHAPTER

5 Conclusion

We focused on the ontology-based query answering (OBQA) problem, which is a fundamental reasoning problem that currently draws attention from both knowledge representation and database communities. The decision version of this problem is the following: given data F, an ontology $\mathcal{R}$, and a (Boolean conjunctive) query q, is that true that q is entailed by F and $\mathcal{R}$? We chose to represent ontologies by means of existential rules. While the mainstream approach to represent ontologies is Description Logics (DLs), the choice of existential rules is motivated by the new light it sheds on OBQA.

The contribution of this dissertation to the field can be divided in two parts. First, we defined a novel class of decidable rules, namely *greedy bounded treewidth sets* of rules (*gbts*) and provided a worst-case optimal algorithm for conjunctive query answering under *gbts* rules. The *gbts* class covers in particular guarded rules, and thus significantly extends lightweight Description Logics such as $\mathcal{EL}$ and DL-Lite, which are the most studied for OBQA. The algorithm we propose is not only worst-case optimal for *gbts*, but also for most of its known subclasses, up to slight adaptations. To achieve such results, we introduced the notion of the *greedy tree decomposition* of a derivation, which is in substance a uniquely defined tree decomposition of the fact associated with a derivation. This greedy tree decomposition being potentially infinite (but of bounded width) in the case of *gbts*, an important challenge was to represent it in a *finite* way. This has been achieved thanks to the definition and the computation of an equivalence relation, making use of a notion of *pattern*. The finite representation we presented is called a *full blocked tree*. A second challenge was to evaluate a query against the computed full blocked tree, in the setting where the full blocked tree is built independently from the query – this allows us to compute this structure offline, which is interesting if several queries are evaluated against the same data and the same ontology. We thus defined a querying operation, called *APT-mapping*, that is sound and complete with respect to the usual semantics. A similar approach has been pursued in the so-called combined approach [Lutz *et al.*, 2009; Kontchakov *et al.*, 2010],

but it has been developed until now only for lightweight description logics. Compared to these languages, we had to face further technical challenges, due to the loss of a strict tree structure and to stronger interactions between rules.

Second, we considered first-order rewritable sets of rules, that is, sets of rules for which a first-order rewriting of any conjunctive query is guaranteed to exist. Most rewriting tools are able to generate such rewritings only under strong restrictions on the ontology. To the best of our knowledge, only one tool [König *et al.*, 2012] is able to generate first-order rewritings, and more precisely, unions of conjunctive queries, for any first-order rewritable set of rules and any conjunctive query. We advocated that a system outputting unions of conjunctive queries is doomed to produce huge rewritings as soon as there are large class or role hierarchies. Unfortunately, this is likely to happen in real-world ontologies, as hierarchies usually are the backbone of such ontologies. We thus proposed an alternative approach, which consists in building rewritings that may use a less restricted form of disjunction, based on the notion of semi-conjunctive queries. We adapted the algorithm from [König *et al.*, 2012], by generalizing the rewriting operation that was defined on conjunctive queries to semi-conjunctive queries. We also distinguished two kinds of rewritings, local and non-local rewritings. Local rewritings introduce disjunctions – this is the main innovative point, and non-local rewritings are a natural recast of the usual operation. We also implemented this adaptation, and provided first experiments showing that using unions of semi-conjunctive queries is beneficial during both the rewriting and the querying steps.

Some questions related to "direct" improvements of this work have already been pointed out in Chapters 3 and 4. More generally, the work presented in this dissertation can be extended in several ways. From a theoretical point of view, a lot remains to be done in order to understand what makes the problem decidable. The distinction that we made between materialization-based and materialization-avoiding approaches is a classical one (though under a variety of other names), but is there a deep theoretical reason to do so? Is it possible to further unify decidability criteria? An argument supporting this possibility is the "equivalence" between rule applications and piece-based rewriting, stating that anything that can be deduced using k rule applications can also be deduced using k-steps of piece-based rewritings [Salvat and Mugnier, 1996]. A possible first move towards this better understanding of decidability properties of OBQA is to further generalize known decidable classes. Several leads can be considered. First, *bts* generalizes *fes*. Can we design a similar generalization of *fus*? A first proposal has been made in [Rudolph and Krötzsch, 2013], by rewriting into more expressive query languages – can we go further? We may also look for sets of rules that are not very expressive, but with good computational properties, and that support combination with transitivity rules or equality rules. We believe that the full blocked tree is a good structure to start with when considering such rule sets. Indeed, we already proposed an adaptation of the algorithm building a full blocked tree to a generalization of $\mathcal{EL}$ that supports predicates of any arity and cyclic dependencies on variables, while staying in the same complexity classes as $\mathcal{EL}$ [Thomazo, 2012]. Complex role inclusion may also be added, provided that some

regularity conditions are observed – the problem being undecidable otherwise [Krötzsch *et al.*, 2007].

Designing new algorithms for OBQA assumes the ability to check whether proposed algorithms are efficient in practice. This requires good benchmarks – and the community is definitely missing publicly available good benchmarks. These should contain a set of test cases containing at least the three following components: an ontology, data mapped with this ontology, and queries expressed within this ontology. Some real-world ontologies are already available, but to the best of our knowledge, all those that belong to *gbts* or are first-order rewritable are also expressible with $\mathcal{EL}$ or DL-Lite. While this could be interpreted as a weak interest for expressive power beyond lightweight description logics in OBQA, we believe that this is mainly due to the lack of tools supporting the design of ontologies by means of existential rules. Furthermore, even when real-world ontologies are available, associated data and queries are, to the best of our knowledge, nonexistent. A first step has been realized towards this by creating Sygenia, which can generate queries and data with an aim of debugging, but publicly available real use cases are definitely missing for now. This problem aside, a much deeper evaluation of COMPACT is needed, using adequate ontologies, queries, and data. A probably difficult but nonetheless interesting question would be to efficiently evaluate semi-conjunctive queries. Concerning the *gbts* algorithm, we do not believe it is readily implementable, mostly because of the querying operation. A rewriting-based mechanism to evaluate queries against a full blocked tree is currently under investigation. The offline part would remain the same, but instead of looking for an APT-mapping of the query, the query would be rewritten into a union of conjunctive queries (or of semi-conjunctive queries). The rewriting would be computed by using the knowledge acquired during the computation of patterns. Last, it would be evaluated on the full blocked tree.

In this dissertation, we focused on the core problem. For instance, we considered only conjunctive queries, with positive atoms. Restricting or extending the query language is also of interest. Even for queries restricted to a single ground atom, entailment remains undecidable. However, query patterns, which allow to consider specific conjunctive queries, have been recently defined [Civili and Rosati, 2012b]. This restriction on queries is then used to rewrite the set of rules into an "equivalent" (to answer the considered queries) set that has better computational properties. More general query languages than conjunctive queries have also been considered, such as Datalog and restrictions of it, such as monadic Datalog [Gottlob and Koch, 2004], as well as regular path queries [Calvanese *et al.*, 2003]. New query languages, with greater expressivity than conjunctive queries but with reasonable complexity of evaluation have been proposed [Rudolph and Krötzsch, 2013]. The design rationale in this latter work was to enhance expressivity while restricting worst-case complexity. A related question it whether the expressivity of current query languages match practical needs. How do we evaluate these needs?

Another way to extend queries (and rules) is to allow for some kind of negation. Non-monotonic negation is arguably useful in practice, *e.g.*, [van Harmelen *et al.*, 2007]. Stratified negation [Apt *et al.*, 1988; Abiteboul *et al.*, 1994] has already been considered

in the framework of guarded rules [Calì *et al.*, 2009]. Stable negation (*i.e.*, based on stable models), which generalizes stratified negation, has been intensively studied in the framework of answer set programming (*e.g.*, [Gelfond and Lifschitz, 1988; Baral, 2003]). By skolemizing existential rules, we obtain a special case of this latter framework, and thus this translation provides us with a potential semantics for existential rules with stable negation. However, defining a direct semantics for answer set programs with existential variables in rule heads without using skolemization does not seem to be trivial. Naturally, classical (monotonic) negation, even restricted to negation on atoms, also raises a lot of problems. Even without any ontology, having negated atoms in data and queries makes the conjunctive query answering problem significantly harder: it jumps from NP-complete to Π_2^p-complete in combined complexity ([Farré *et al.*, 2007], for the problem of inclusion of conjunctive queries with negation). This is due to a combinatorial explosion – each atom which is not explicitly stated as true or false, gives rise to a choice. Moreover, when rules are added, the notion of "applying a rule" is not obvious anymore. In [Mugnier *et al.*, 2012], we considered the core problem, *i.e.*, without ontologies (or very restricted ontologies, that is, concept or relation hierarchies), and defined parameters that make the complexity of the problem drop when their value is fixed.

In this dissertation, we also assume the ontology to be consistent. This may be a reasonable assumption, since ontologies are usually developed by domain experts that reached a consensus. Moreover, (semi-)automatic debugging tools are available, which help to discover and fix problems at the ontological level. Last, an inconsistency in an ontology plausibly originates from a formalization problem, and it seems reasonable to fix this problem before using the ontology. As for data, doing such a consistency assumption does not seem reasonable at all, since data could come from multiple sources, and their authors may not be experts. There could thus be inconsistencies between the data and the ontology. The semantics of first-order logic states that everything is entailed by an inconsistent knowledge base. This behavior is not satisfying for the query answering task. The notion of *repair* (intuitively, a consistent subset of the database as close as possible to the original one) has been introduced in the database community (see [Chomicki, 2007] for a survey). Recently, semantics that allow to make reasonable inferences even with inconsistent knowledge bases have been proposed [Lembo *et al.*, 2010; Rosati, 2011; Bienvenu, 2012; Lembo *et al.*, 2012; Lukasiewicz *et al.*, 2012]. Since consistent query answering of DL-Lite ontologies is already complex, the challenge is even greater for (decidable) classes of existential rules.

How can our methods, and in particular the full blocked tree, be used when introducing these extensions? This question remains open.

Index

Bibliography

S. Abiteboul, R. Hull and V. Vianu : *Foundations of Databases*. Addison Wesley, 1994. Cited pages 4, 16, and 111.

A. Acciarri, D. Calvanese, G. De Giacomo, D. Lembo, M. Lenzerini, M. Palmieri and R. Rosati : Quonto: Querying ontologies. *In AAAI*, pages 1670–1671, 2005. Cited page 107.

H. Andréka, I. Németi and J. van Benthem : Modal Languages and Bounded Fragments of Predicate Logic. *J. of Philosogical Logic*, 27:217–274, 1998. Cited page 32.

K. R. Apt, H. A. Blair and A. Walker : Towards a theory of declarative knowledge. *In Foundations of Deductive Databases and Logic Programming.*, pages 89–148, 1988. Cited page 111.

F. Baader : Terminological cycles in a description logic with existential restrictions. *In IJCAI*, pages 325–330, 2003. Cited page 14.

F. Baader, S. Brandt and C. Lutz : Pushing the $\mathcal{EL}$ envelope. *In IJCAI*, pages 364–369, 2005. Cited page 15.

F. Baader, D. Calvanese, D. McGuinness, D. Nardi and P. Patel-Schneider, éditeurs. *The Description Logic Handbook: Theory, Implementation, and Applications*. Cambridge University Press, second édition, 2007. Cited page 12.

F. Baader, C. Lutz and A.-Y. Turhan : Small is again beautiful in description logics. *KI*, 24(1):25–33, 2010. Cited page 14.

J.-F. Baget : Improving the forward chaining algorithm for conceptual graphs rules. *In KR'04*, pages 407–414. AAAI Press, 2004. Cited pages 37 and 44.

J.-F. Baget : Private communication, 2012. Cited page 83.

J.-F. Baget, M. Leclère and M.-L. Mugnier : Walking the Decidability Line for Rules with Existential Variables. *In KR*. AAAI Press, 2010. Cited page 33.

J.-F. Baget, M. Leclère, M.-L. Mugnier and E. Salvat : Extending Decidable Cases for Rules with Existential Variables. *In IJCAI*, pages 677–682, 2009. Cited pages 21, 33, and 37.

J.-F. Baget, M. Leclère, M.-L. Mugnier and E. Salvat : On Rules with Existential Variables: Walking the Decidability Line. *Artif. Intell.*, 175(9-10):1620–1654, 2011a. Cited pages 16, 18, 21, 23, 25, 31, 35, and 41.

J.-F. Baget, M.-L. Mugnier, S. Rudolph and M. Thomazo : Walking the complexity lines for generalized guarded existential rules. *In IJCAI*, pages 712–717, 2011b. Cited pages 43 and 44.

J.-F. Baget, M.-L. Mugnier and M. Thomazo : Towards farsighted dependencies for existential rules. *In RR*, pages 30–45, 2011c. Cited pages 38 and 44.

C. Baral : *Knowledge Representation, Reasoning, and Declarative Problem Solving*. Cambridge University Press, New York, NY, USA, 2003. ISBN 0521818028. Cited page 112.

C. Beeri and M.Y. Vardi : A Proof Procedure for Data Dependencies. *Journal of the ACM*, 31(4):718–741, 1984. Cited page 18.

M. Bienvenu : Inconsistency-tolerant conjunctive query answering for simple ontologies. *In Description Logics*, 2012. Cited page 112.

A. Calì, G. Gottlob and M. Kifer : Taming the Infinite Chase: Query Answering under Expressive Relational Constraints. *In KR*, pages 70–80, 2008. Cited pages 16, 21, 31, 32, 33, and 43.

A. Calì, G. Gottlob and T. Lukasiewicz : A General Datalog-Based Framework for Tractable Query Answering over Ontologies. *In PODS*, pages 77–86. ACM, 2009. ISBN 978-1-60558-553-6. URL http://doi.acm.org/10.1145/1559795.1559809. Cited pages 16, 18, 23, 31, 43, 83, and 112.

A. Calì, G. Gottlob and T. Lukasiewicz : Datalog extensions for tractable query answering over ontologies. *In* Roberto de Virgilio, Fausto Giunchiglia and Letizia Tanca, éditeurs : *Semantic Web Information Management*, pages 249–279. Springer Berlin Heidelberg, 2010. ISBN 978-3-642-04328-4. URL http://dx.doi.org/10.1007/978-3-642-04329-1_12. Cited page 43.

A. Calì, G. Gottlob and T. Lukasiewicz : A general datalog-based framework for tractable query answering over ontologies. *J. Web Sem.*, 14, 2012. Cited page 4.

A. Calì, G. Gottlob and A. Pieris : Query answering under non-guarded rules in datalog+/-. *In RR*, pages 1–17, 2010a. Cited page 43.

A. Calì, G. Gottlob and A. Pieris : Query rewriting under non-guarded rules. *In AMW*, 2010b. Cited pages 34 and 40.

D. Calvanese, G. De Giacomo, D. Lembo, M. Lenzerini and R. Rosati : Dl-lite: Tractable description logics for ontologies. *In AAAI*, pages 602–607, 2005. Cited page 15.

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. Cited pages 87 and 106.

D. Calvanese, G. De Giacomo, M. Lenzerini and M. Y. Vardi : Reasoning on regular path queries. *SIGMOD Record*, 32(4):83–92, 2003. Cited page 111.

A. K. Chandra, H. R. Lewis and J. A. Makowsky : Embedded implicational dependencies and their inference problem. *In STOC*, pages 342–354, 1981. Cited page 43.

M. Chein and M.-L. Mugnier : *Graph-based Knowledge Representation: Computational Foundations of Conceptual Graphs*. Springer Publishing Company, Incorporated, 1 édition, 2008. ISBN 1848002858, 9781848002852. Cited page 4.

J. Chomicki : Consistent query answering: Five easy pieces. *In ICDT*, pages 1–17, 2007. Cited page 112.

A. Chortaras, D. Trivela and G. B. Stamou : Optimized query rewriting for OWL 2 QL. *In CADE*, pages 192–206, 2011. Cited page 107.

C. Civili and R. Rosati : A broad class of first-order rewritable tuple-generating dependencies. *In Datalog*, pages 68–80, 2012a. Cited page 40.

C. Civili and R. Rosati : Query patterns for existential rules. *In RR*, pages 42–57, 2012b. Cited page 111.

B. Courcelle : The Monadic Second-Order Logic of Graphs: I. Recognizable Sets of Finite Graphs. *Information and Computation*, 85(1):12–75, 1990. Cited page 22.

E. Dantsin, T. Eiter, G. Gottlob and A. Voronkov : Complexity and expressive power of logic programming. *ACM Comput. Surv.*, 33(3):374–425, 2001. Cited page 43.

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. Cited pages 35, 36, and 43.

C. Farré, W. Nutt, E. Teniente and T. Urpí : Containment of conjunctive queries over databases with null values. *In ICDT*, pages 389–403, 2007. Cited page 112.

M. Gelfond and V. Lifschitz : The stable model semantics for logic programming. *In ICLP/SLP*, pages 1070–1080, 1988. Cited page 112.

G. Gottlob and C. Koch : Monadic datalog and the expressive power of languages for web information extraction. *J. ACM*, 51(1):74–113, 2004. Cited page 111.

G. Gottlob, G. Orsi and A. Pieris : Ontological queries: Rewriting and optimization. *In ICDE*, pages 2–13, 2011. Cited page 107.

G. Gottlob and T. Schwentick : Rewriting ontological queries into small nonrecursive datalog programs. *In KR*, 2012. Cited pages 23 and 108.

B. Cuenca Grau, I. Horrocks, M. Krötzsch, C. Kupke, D. Magka, B. Motik and Z. Wang : Acyclicity conditions and their application to query answering in description logics. *In KR*, 2012. Cited pages 37, 39, and 43.

M. Imprialou, G. Stoilos and B. Cuenca Grau : Benchmarking ontology-based query rewriting systems. *In AAAI*, 2012. Cited pages 103, 104, and 108.

S. Kikot, R. Kontchakov and M. Zakharyaschev : Polynomial conjunctive query rewriting under unary inclusion dependencies. *In RR*, pages 124–138, 2011. Cited pages 43 and 107.

M. König, M. Leclère, M.-L. Mugnier and M. Thomazo : A sound and complete backward chaining algorithm for existential rules. *In RR*, pages 122–138, 2012. Cited pages 6, 25, 30, 44, 103, 107, and 110.

M. König, M. Leclère, M.-L. Mugnier and M. Thomazo : On the exploration of the query rewriting space with existential rules. *In RR*, 2013. Cited page 30.

R. Kontchakov, C. Lutz, D. Toman, F. Wolter and M. Zakharyaschev : The combined approach to query answering in dl-lite. *In KR*, 2010. Cited pages 83 and 109.

R. Kontchakov, C. Lutz, D. Toman, F. Wolter and M. Zakharyaschev : The combined approach to ontology-based data access. *In IJCAI*, pages 2656–2661, 2011. Cited page 83.

M. Krötzsch : Private communication, 2012. Cited page 39.

M. Krötzsch and S. Rudolph : Extending decidable existential rules by joining acyclicity and guardedness. *In IJCAI*, pages 963–968, 2011. Cited pages 37, 40, and 43.

M. Krötzsch, S. Rudolph and P. Hitzler : Conjunctive queries for a tractable fragment of OWL 1.1. *In ISWC/ASWC*, pages 310–323, 2007. Cited pages 43 and 111.

M. Leclère, M.-L. Mugnier and S. Rocher : KIABORA: an analyzer of existential rule bases. *In RR*, 2013. Cited page 41.

D. Lembo, M. Lenzerini, R. Rosati, M. Ruzzi and D. F. Savo : Inconsistency-tolerant semantics for description logics. *In RR*, pages 103–117, 2010. Cited page 112.

D. Lembo, M. Lenzerini, R. Rosati, M. Ruzzi and D. F. Savo : Inconsistency-tolerant first-order rewritability of dl-lite with identification and denial assertions. *In Description Logics*, 2012. Cited page 112.

N. Leone, M. Manna, G. Terracina and P. Veltri : Efficiently computable datalog; programs. *In KR*, 2012. Cited pages 33, 42, and 43.

T. Lukasiewicz, M. V. Martinez and G. I. Simari : Inconsistency handling in datalog+/- ontologies. *In ECAI*, 2012. Cited page 112.

C. Lutz, I. Seylan, D. Toman and F. Wolter : The combined approach to OBDA: Taming role hierarchies using filters. *In SSWS+HPCSW*, 2012. Cited page 103.

C. Lutz, D. Toman and F. Wolter : Conjunctive Query Answering in the Description Logic $\mathcal{EL}$ Using a Relational Database System. *In IJCAI*, pages 2070–2075, 2009. Cited pages 83 and 109.

B. Marnette : Generalized schema-mappings: from termination to tractability. *In PODS*, pages 13–22, 2009. Cited page 37.

M.-L. Mugnier, G. Simonet and M. Thomazo : On the complexity of entailment in existential conjunctive first-order logic with atomic negation. *Inf. Comput.*, 215:8–31, 2012. Cited pages 44 and 112.

H. Pérez-Urbina, I. Horrocks and B. Motik : Efficient query answering for OWL 2. *In International Semantic Web Conference*, pages 489–504, 2009. Cited pages 103 and 104.

N. Robertson and P. D. Seymour : Graph minors. ii. algorithmic aspects of tree-width. *J. Algorithms*, 7(3):309–322, 1986. Cited page 10.

M. Rodriguez-Muro and D. Calvanese : High performance query answering over DL-lite ontologies. *In KR*, 2012. Cited pages 89 and 103.

R. Rosati : On the complexity of dealing with inconsistency in description logic ontologies. *In IJCAI*, pages 1057–1062, 2011. Cited page 112.

R. Rosati and A. Almatelli : Improving query answering over dl-lite ontologies. *In KR*, 2010. Cited pages 23 and 108.

S. Rudolph : Private communication, 2012. Cited page 83.

S. Rudolph and M. Krötzsch : Flag & check: Data access with monadically defined queries. *In Proceedings of the 32nd ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems (PODS'13)*. ACM, 2013. to appear. Cited pages 23, 108, 110, and 111.

E. Salvat and M.-L. Mugnier : Sound and complete forward and backward chainingd of graph rules. *In ICCS*, pages 248–262, 1996. Cited pages 16, 25, 27, and 110.

J. F. Sowa : *Conceptual Structures: Information Processing in Mind and Machine*. Addison-Wesley, 1984. ISBN 0-201-14472-7. Cited page 4.

G. Stefanoni, B. Motik and I. Horrocks : Small datalog query rewritings for EL. *In Description Logics*, 2012. Cited page 23.

M. Thomazo : From EL to tractable existential rules with complex role inclusions. *In Description Logics*, 2012. Cited pages 44 and 110.

M. Thomazo : Compact rewriting for existential rules. *In IJCAI*, 2013. Cited page 44.

M. Thomazo, J.-F. Baget, M.-L. Mugnier and S. Rudolph : A generic querying algorithm for greedy sets of existential rules. *In KR*, 2012. Cited page 44.

F. van Harmelen, V. Lifschitz and B. Porter : *Handbook of Knowledge Representation*. Elsevier Science, San Diego, USA, 2007. ISBN 0444522115, 9780444522115. Cited page 111.

T. Venetis, G. Stoilos and G. B. Stamou : Incremental query rewriting for OWL 2 QL. *In Description Logics*, 2012. Cited pages 104 and 107.

Abstract

Ontology-based data access (OBDA) aims at enriching query answering by taking general background knowledge into account when evaluating queries. This background knowledge is represented by means of an ontology, that is expressed in this thesis by a very expressive class of first-order formulas, called existential rules (sometimes also tuple-generating dependencies and Datalog+/-). The high expressivity of the used formalism results in the undecidability of query answering, and numerous decidable classes (that is, restrictions on the sets of existential rules) have been proposed in the literature. The contribution of this thesis is two-fold: first, we propose a unified view of a large part of these classes, together with a complexity analysis and a worst-case optimal algorithm for the introduced generic class. Second, we consider the popular approach of query rewriting, and propose a generic algorithm that overcomes trivial causes of combinatorial explosion that make classical approaches inapplicable.

Keywords: *Artificial Intelligence, Knowledge representation and reasoning, Datalog +/-, Existential rules, Conjunctive queries*

Résumé

L'objectif du problème appelé "Ontology-based data access" (OBDA) est d'améliorer la réponse à des requêtes en prenant en compte des connaissances d'ordre général durant l'évaluation des requêtes. Ces connaissances générales sont représentées à l'aide d'une ontologie, qui est exprimée dans cette thèse grâce à des formules logiques du premier ordre, appelées règles existentielles, et aussi connues sous le nom de "tuple-generating dependencies" et Datalog+/-. L'expressivité des formules utilisées est telle que l'évaluation de requêtes devient un problème indécidable, et cela a conduit la communauté à définir de nombreux cas décidables, c'est-à-dire des restrictions sur les ensembles de règles existentielles considérés. La contribution de cette thèse est double : tout d'abord, nous proposons une vue unifiée sur une grande fraction des cas décidables connus, et fournissons par là même une analyse de complexité et un algorithme optimal dans le pire des cas. Nous considérons également l'approche couramment utilisée de réécriture de requêtes, et proposons un algorithme générique qui permet de surmonter certaines causes évidentes d'explosion combinatoire qui rendent les approches classiques pratiquement inapplicables.

Mots clefs : *Intelligence Artificielle, Représentation des connaissances et raisonnement, Datalog+/-, Règles existentielles, Requêtes conjonctives*

LIRMM — 161, rue Ada — 34095 Montpellier cedex 5 — France