Worst-case Optimal Query Answering for Greedy Sets of Existential Rules and Their Subclasses

Sebastian Rudolph, Michael Thomazo, Jean-François Baget, Marie-Laure Mugnier

Worst-case Optimal Query Answering for Greedy Sets of Existential Rules and Their Subclasses

Sebastian Rudolph[*] SEBASTIAN.RUDOLPH@TU-DRESDEN.DE
TU Dresden, Germany

Michaël Thomazo[*] MICHAEL.THOMAZO@TU-DRESDEN.DE
TU Dresden, Germany

Jean-François Baget BAGET@LIRMM.FR
Inria, France

Marie-Laure Mugnier MUGNIER@LIRMM.FR
University Montpellier 2, France

Abstract

The need for an ontological layer on top of data, associated with advanced reasoning mechanisms able to exploit the semantics encoded in ontologies, has been acknowledged both in the database and knowledge representation communities. We focus in this paper on the ontological query answering problem, which consists of querying data while taking ontological knowledge into account. More specifically, we establish complexities of the conjunctive query entailment problem for classes of *existential rules* (also called tuple-generating dependencies, Datalog$^{\pm}$ rules, or $\forall\exists$-rules). Our contribution is twofold. First, we introduce the class of *greedy bounded-treewidth sets* (*gbts*) of rules, which covers guarded rules, and their most well-known generalizations. We provide a generic algorithm for query entailment under *gbts*, which is worst-case optimal for combined complexity with or without bounded predicate arity, as well as for data complexity and query complexity. Secondly, we classify several *gbts* classes, whose complexity was unknown, with respect to combined complexity (with both unbounded and bounded predicate arity) and data complexity to obtain a comprehensive picture of the complexity of existential rule fragments that are based on diverse guardedness notions. Upper bounds are provided by showing that the proposed algorithm is optimal for all of them.

∗. This work has been partially realized while S. Rudolph was working at AIFB, KIT, Germany and M. Thomazo was a Ph.D. student at University Montpellier 2

1. Introduction

Intelligent methods for searching and managing large amounts of data require rich and elaborate schematic "ontological" knowledge. The need for an ontological layer on top of that data, associated with advanced querying mechanisms able to exploit the semantics encoded in ontologies[1], has been widely acknowledged both in the knowledge representation (KR) and database communities.

Deductive databases and KR typically adopt two different perspectives on how to add this ontological layer to the picture of plain query answering (cf. Rudolph, 2014). In deductive databases, this knowledge is considered part of the query, forming a so-called ontology-mediated query to be executed on the database. According to the KR perspective, knowledge is encoded in an ontology, and queries are asked to a knowledge base composed of the data and the ontology. In this paper, we will focus on the KR perspective.

Indeed, ontologies, which typically encode general knowledge about the domain of interest, can be used to infer data that are not explicitely stored, hence palliating incompleteness in databases (Calì, Lembo, & Rosati, 2003). They can also be used to enrich the vocabulary of data sources, which allows a user to abstract from the specific way data are stored. Finally, when several data sources use different vocabularies, ontologies can be used to align these vocabularies.

Given a knowledge base (KB) composed of an ontology and of factual knowledge, and a query, the ontology-based query answering problem consists in computing the set of answers to the query on the KB, while taking implicit knowledge represented in the ontology into account. We make here the simplifying assumption that the ontology and the database use the same vocabulary. Otherwise, mappings have to be defined between both vocabularies, as in the ontology-based data access framework (Poggi, Lembo, Calvanese, De Giacomo, Lenzerini, & Rosati, 2008). As most work in this area, we focus on conjunctive queries (CQs), the basic and most frequent querying formalism in databases.

In the Semantic Web area, one of the most prominent fields where KR technology is practically applied, ontological knowledge is often represented by means of formalisms based on description logics (DLs, (Baader, Calvanese, McGuinness, Nardi, & Patel-Schneider, 2007; Rudolph, 2011)). However, DLs are restricted in terms of expressivity in that they usually support only unary and binary predicates and that terminological expressiveness is essentially restricted to tree-like dependencies between the atoms of a formula. Moreover, DL research has traditionally been focusing on so-called standard reasoning tasks about the knowledge base, which are reducible to knowledge base satisfiability, for instance classifying concepts; querying tasks were essentially restricted to ground atom entailment. Answering full conjunctive queries (CQs) over DL knowledge bases has become a subject of research only recently[2], turning out to be extremely complex (e.g., for the classical DL $\mathcal{ALCI}$, it is already 2ExpTime-complete, and still NP-complete in the size of the data). Consequently, conjunctive query answering has been particularly studied on less expressive DLs, such as DL-Lite (Calvanese, De Giacomo, Lembo, Lenzerini, & Rosati, 2007) and $\mathcal{EL}$ (Baader, 2003; Lutz, Toman, & Wolter, 2009). These DLs are the basis of so-called tractable profiles

1. In this paper, we reserve the term *ontology* to general domain knowledge–sometimes also called *terminological* knowledge–in order to clearly distinguish it from the factual data–or *assertional* knowledge.
2. CQ answering in the context of DLs was first mentioned by Levy and Rousset (1996), with the first publication focusing on that subject by Calvanese, De Giacomo, and Lenzerini (1998).

OWL 2 QL and OWL 2 EL of the Semantic Web language OWL 2 (W3C OWL Working Group, 2009; Motik, Cuenca Grau, Horrocks, Wu, Fokoue, & Lutz, 2009a).[3]

On the other hand, querying large amounts of data is the fundamental task in databases. Therefore, the challenge in this domain is now to access data while taking ontological knowledge into account. The deductive database language Datalog allows to express some ontological knowledge. However, in Datalog rules, variables are range-restricted, i.e., all variables in the rule are universally quantified, which does not allow to infer the existence of initially unknown domain individuals (a capability called *value invention* in databases (Abiteboul, Hull, & Vianu, 1994)). Yet, this feature has been recognized as crucial in an open-world perspective, where it cannot be assumed that all individuals are known in advance.

To accommodate the requirements sketched above – value invention and complex relationships – we consider here an extension of first-order function-free Horn rules that allows for existentially quantified variables in the rule heads and thus features value invention. More precisely, these extended rules are of the form $Body \rightarrow Head$, where *Body* and *Head* are conjunctions of atoms, and variables occurring only in the *Head* are existentially quantified, hence their name "existential rules" in (Baget, Mugnier, Rudolph, & Thomazo, 2011; Krötzsch & Rudolph, 2011).

Example 1. *Consider the existential rule*

$$R = \forall x\Big(human(x) \rightarrow \exists y\big(hasParent(x,y) \wedge human(y)\big)\Big)$$

and a fact $F = human(a)$*, where* a *is a constant. The application of* R *to* F *produces new factual knowledge, namely*

$$\exists y_0\big(hasParent(a,y_0) \wedge human(y_0)\big),$$

where y_0 *is a variable denoting an unknown individual. Note that* R *could be applied again to* $human(y_0)$*, which would lead to create another existentially quantified variable, and so on.*

Such rules are well-known in databases as Tuple-Generating Dependencies (TGDs) (Beeri & Vardi, 1984) and have been extensively used, e.g., for data exchange (Fagin, Kolaitis, Miller, & Popa, 2005). Recently, the corresponding logical fragment has gained new interest in the context of ontology-based query answering. It has been introduced as the Datalog$^{\pm}$ framework in (Calì, Gottlob, & Kifer, 2008; Calì, Gottlob, & Lukasiewicz, 2009; Calì, Gottlob, Lukasiewicz, Marnette, & Pieris, 2010), and independently, stemming from graph-based knowledge representation formalisms (Chein & Mugnier, 2009), as $\forall\exists$ rules (Baget, Leclère, Mugnier, & Salvat, 2009; Baget, Leclère, & Mugnier, 2010).

This rule-based framework generalizes the core of the lightweight description logics mentioned above, namely DL-Lite and $\mathcal{EL}$.[4] Moreover, in the case of the DL-Lite family (Cal-

3. Beside the profiles based on DL-Lite and $\mathcal{EL}$, there is a third OWL 2 tractable profile, OWL 2 RL, which can be seen as a restriction of Datalog.

4. The DL constructor called existential restriction ($\exists R.C$) is fundamental in these DLs. The logical encoding of an axiom that contains $\exists R.C$ in its right-hand side requires an existentially quantified variable in the corresponding rule head. For instance, the rule from Example 1 can be seen as the logical translation of the DL axiom $Human \sqsubseteq \exists hasParent.Human$.

vanese et al., 2007), it has been shown that this covering by a Datalog$^{\pm}$ fragment is done without increasing complexity (Calì et al., 2009).

Several fundamental decision problems can be associated with conjunctive query answering under existential rules. In this paper, we consider the entailment problem of a Boolean conjunctive query under existential rules, which we are now able to define formally. A Boolean conjunctive query is an existentially closed conjunction of (function-free) atoms. A set of facts has the same form. A knowledge base is composed of a set of facts and a set of existential rules. The entailment problem takes as input a knowledge base and a Boolean conjunctive query and asks if this query is entailed by the knowledge base.

The presence of existentially quantified variables in rule heads, associated with arbitrarily complex conjunctions of atoms, makes the entailment problem undecidable (Beeri & Vardi, 1981; Chandra, Lewis, & Makowsky, 1981a). Since the birth of TGDs, and recently within the Datalog$^{\pm}$ and $\forall\exists$ rule frameworks, various conditions of decidability have been exhibited. Three "abstract" classes have been introduced in (Baget et al., 2010) to describe known decidable behaviors: an obvious condition of decidability is the finiteness of the forward chaining (known as the *chase* in the TGD framework (Maier, Mendelzon, & Sagiv, 1979; Johnson & Klug, 1984; Beeri & Vardi, 1984)); sets of rules ensuring this condition are called *finite expansion sets* (***fes***); a more general condition introduced in (Calì et al., 2008) accepts infinite forward chaining provided that the facts generated have a bounded treewidth (when seen as graphs); such sets of rules are called *bounded-treewidth sets* (***bts***); then decidability follows from the decidability of first-order logic (FOL) classes with the bounded-treewidth model property (Courcelle, 1990). The third condition, giving rise to *finite unification sets* (***fus***), relies on the finiteness of (a kind of) backward chaining mechanism, this condition is also known as *first-order rewritability*. None of these three abstract classes is recognizable, i.e., the problem of deciding whether a given set of rules is ***fes***, ***bts***, or ***fus*** is undecidable (Baget et al., 2010).

In this paper, we focus on the ***bts*** paradigm and its main "concrete" classes. (Pure) *Datalog* rules (i.e., without existential variables) are ***fes*** (thus ***bts***). *Guarded* (***g***) rules (Calì et al., 2008) are inspired by the guarded fragment of FOL (Andréka, van Benthem, & Németi, 1996; Andréka, Németi, & van Benthem, 1998). Their body has an atom, called a guard, that contains all variables from the body. Guarded rules are ***bts*** (and not ***fes***). They are generalized by *weakly guarded rules* (***wg***), in which the guarding condition is relaxed: only so-called "affected" variables need to be guarded; intuitively, affected variables are variables that are possibly mapped, during the forward chaining process, to newly created variables (Calì et al., 2008). ***wg*** rules include Datalog rules (in which there are no affected variables). Other decidable classes rely on the notion of the *frontier* of a rule (the set of variables shared between the body and the head of a rule). In a *frontier-one* rule (***fr1***), the frontier is restricted to a single variable (Baget et al., 2009). In a frontier-guarded rule (***fg***), an atom in the body guards the frontier (Baget et al., 2010). Hence, ***fg*** rules generalize both *guarded* rules and ***fr1*** rules. When requiring only affected variables from the frontier to be guarded, we obtain the still decidable class of *weakly frontier-guarded rules* (***wfg***), which generalizes both ***fg*** and ***wg*** classes (Baget et al., 2010). Not considered until now were rule sets obtained by straightforward combinations of the above properties: *guarded frontier-one rules* (***gfr1***) as well as *weak frontier-one rules* (***wfr1***) and *weakly guarded frontier-one rules*

(*wgfr1*). Table 1 summarizes the considered existential rule fragments with respect to their constraints on frontier variables and guardedness.

syntactic properties / **class (abbrv)**	non-affected variables must be guarded	non-frontier variables must be guarded	at most one frontier variable that needs guarding
guarded frontier-one rules (*gfr1*)	yes	yes	yes
guarded rules (*g*)	yes	yes	no
frontier-one rules (*fr1*)	yes	no	yes
frontier-guarded rules (*fg*)	yes	no	no
weakly guarded frontier-one rules (*wgfr1*)	no	yes	yes
weakly guarded rules (*wg*)	no	yes	no
weakly frontier-one rules (*wfr1*)	no	no	yes
weakly frontier-guarded rules (*wfg*)	no	no	no

Table 1: Considered classes with syntactic properties.

Example 2. *We consider the following relations, where the subscripts indicate the arity of the relation:* $\text{project}_{/3}$, $\text{projectField}_{/2}$, $\text{projectDpt}_{/2}$, $\text{hasManager}_{/2}$, $\text{memberOf}_{/2}$, $\text{isSensitiveField}_{/1}$ *and* $\text{isCriticalManager}_{/1}$. *Intuitively,* $\text{project}(x, d, z)$ *means that* x *is a project in department* d *and is about field* z*; relations* projectField *and* projectDpt *are projections of the* project *relation;* $\text{hasManager}(x, y)$ *and* $\text{member}(y, d)$ *respectively mean that* x *is managed by* y *and* y *is member of* d*; relations* isSensitiveField *and* isCriticalManager *respectively apply to fields and managers. Let* $\mathcal{R}$ *be the following set of existential rules built on this vocabulary:*

- Decomposition of the relation project into two binary relations
  $R_0 = \forall x \forall d \forall z (project(x, d, z) \rightarrow projectDpt(x, d) \wedge projectField(x, z)$
- "Every project has a manager"
  $R_1 = \forall x \forall z (projectField(x, z) \rightarrow \exists y \; hasManager(x, y))$
- "Every managed project has some field"
  $R_2 = \forall x \forall y (hasManager(x, y) \rightarrow \exists z \; projectField(x, z))$
- "The manager of a project is a member of the department that owns the project"
  $R_3 = \forall x \forall y \forall d (hasManager(x, y) \wedge projectDpt(x, d) \rightarrow memberOf(y, d))$
- "If a manager manages a project in a sensitive field, then (s)he is a critical manager"
  $R_4 = \forall x \forall y \forall z (hasManager(x, y) \wedge projectField(x, z) \wedge isSensitiveField(z) \rightarrow isCriticalManager(y))$
- "Every critical manager manages a project in a sensible field"
  $R_5 = \forall y (isCriticalManager(y) \rightarrow \exists x \exists z (hasManager(x, y) \wedge projectField(x, z) \wedge isSensitiveField(z)))$

Note that rules R_0, R_3 and R_4 do not introduce any existential variable. Rules R_0, R_1, R_2 and R_5 have an atomic body, hence they are trivially guarded. Rule R_4 is not guarded, but it is frontier-one. Hence, if we exclude R_3, all rules are frontier-guarded. R_3 is not frontier-guarded, since no atom from the body contains both frontier variables y and d; however, we remark that variable d is not affected, i.e., it can never be mapped to a newly created variable (indeed, the only rule able to produce an atom with predicate projectDpt *is R_0; the variables in this atom come from the body atom with predicate* project*, which never appears in a rule head, hence can only be mapped to initially present data). Affected frontier-variables are guarded in all rules, hence $\mathcal{R}$ is weakly frontier-guarded. $\mathcal{R}$ is even weakly frontier-one, since for each rule the frontier contains at most one affected variable.*

Contrarily to *fes* and *fus*, the definition of *bts* is not based on a constructive entailment procedure. The complexity of the subclasses *g* and *wg* is known and an algorithm for the corresponding entailment problem has been provided (Calì et al., 2008, 2009). However, this is not the case for the classes *gfr1*, *fr1*, *fg*, *wgfr1*, *wfr1*, *wfg*, and *gbts*. The aim of this paper is to solve these algorithmic and complexity issues.

Our contribution is threefold. First, by imposing a restriction on the allowed forward-chaining derivation sequences, we define a subclass of *bts*, namely *greedy bounded-treewidth sets* of rules (*gbts*), which have the nice property of covering the *wfg* class. *gbts* are defined by a rather simple condition: when such a set is processed in forward chaining and a rule R is applied, all frontier variables of R which are not mapped to terms from the initial data set must be uniformly mapped to terms introduced by one *single* previous rule application. The fundamental property satisfied thanks to this condition is that any derivation can be naturally associated with a bounded-width tree decomposition of the derived facts, which can be built in a "greedy manner", that is, on the fly during the forward chaining process. We also prove that *wfg* and *gbts* have essentially the same expressivity.

Secondly, we provide a generic algorithm for the *gbts* class, which is worst-case optimal for data complexity, for combined complexity (with or without bound on the arity of involved predicates), and for query complexity. We furthermore show that this algorithm can be slightly adapted to be worse-case optimal for subclasses with smaller complexities.

Thirdly, we classify *gfr1*, *fr1*, *fg*, *wgfr1*, *wfr1*, *wfg*, and *gbts* with respect to both combined (with and without predicate arity bound) and data complexities. We also consider the case of rules with an acyclic (more precisely, hypergraph-acyclic) body and point out that body-acyclic *fg* rules coincide with guarded rules from an expressivity and complexity perspective.

Fig. 1 shows the complexity lines for these classes of rules with three complexity measures, namely combined complexity without or with bound on the predicate arity, and data complexity. Notice in particular that the two direct extensions of guarded rules, i.e., weakly guarded and frontier-guarded rules, do not behave in the same way with respect to the different complexity measures: for data complexity, *fg* rules remain in PTime, while *wg* rules are in ExpTime; on the other hand, for bounded-arity combined complexity, *fg* rules are in 2ExpTime, while *wg* rules remain in ExpTime. Precise complexity results obtained are given in Tab. 1. All complexities are complete for their class. New results are indicated by a star.

Paper Organization In Section 2, basic definitions and results about existential rules are recalled. Section 3 introduces the *gbts* class and specifies its relationships with the *wfg*

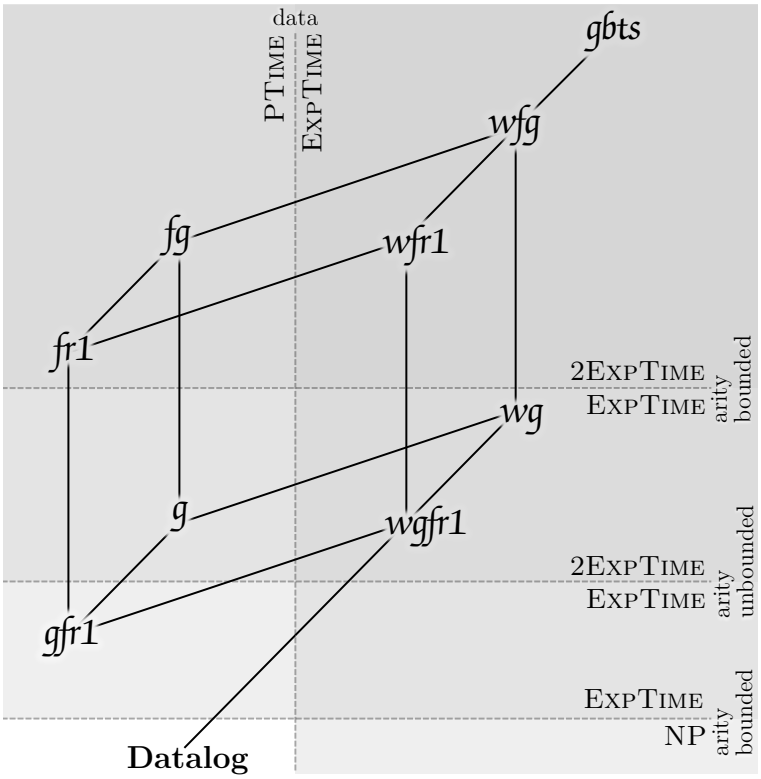

Figure 1: Existential rule fragments, their relative expressiveness, and complexities.

Class	**arity unbounded**	**arity bounded**	**Data Complexity**
Datalog	ExpTime	NP	PTime
gfr1	ExpTime*	ExpTime*	PTime*
g	2ExpTime	ExpTime	PTime
fr1	2ExpTime*	2ExpTime*	PTime*
fg	2ExpTime*	2ExpTime*	PTime*
wgfr1	2ExpTime*	ExpTime*	ExpTime*
wg	2ExpTime	ExpTime	ExpTime
wfr1	2ExpTime*	2ExpTime*	ExpTime*
wfg	2ExpTime*	2ExpTime*	ExpTime*
gbts	2ExpTime*	2ExpTime*	ExpTime*

Table 2: Combined and Data Complexities

class. Section 4 is devoted to an algorithm for entailment with *gbts* and to the associated complexity results. The next sections consider increasingly simpler classes, namely *fg* and *fr1* (Section 5) and body-acyclic *fg*/*fr1* rules (Section 6); several reductions are provided,

which provide tight lower bounds and allow to completely classify these classes for data and combined complexities. Related work is reviewed in Section 7.

This article is an extended version of two papers published at IJCAI 2011 (Baget et al., 2011) and KR 2012 (Thomazo, Baget, Mugnier, & Rudolph, 2012), respectively. It provides detailed proofs of the results presented in these conference papers and benefits from further clarifications concerning the *gbts* algorithm, stemming from Michaël Thomazo's PhD thesis (Thomazo, 2013). Furthermore, it contains complexity results for new classes of rules which complement the picture, namely *gfr1*, *wgfr1* and *wfr1*.

2. Preliminaries

We assume the reader to be familiar with syntax and semantics of first-order logic. We do not consider functional symbols except constants, hence a term is simply a variable or a constant. An *atom* is thus of the form $p(t_1, \ldots, t_k)$ where p is a predicate with arity k, and the t_i are terms. If not otherwise specified, a conjunction is a finite conjunction of atoms. We denote it by $C[\mathbf{x}]$, where $\mathbf{x}$ is the set of variables occurring in C.

A *fact* is the existential closure of a conjunction.[5] A Boolean *conjunctive query* (CQ) has the same form as a fact. We may also represent conjunctions of atoms, facts, and CQs as sets of atoms. Given an atom or a set of atoms A, we denote by $vars(A)$ the set of variables, and by $terms(A)$ the set of terms, occurring in A. Given conjunctions F and Q, a *homomorphism* π from Q to F is a substitution of $vars(Q)$ by $terms(F)$ such that $\pi(Q) \subseteq F$ (we say that Q *maps* to F by π). For convenience, we often assume the domain of π extended to $terms(Q)$, by mapping constants to themselves. Given an atom $a = p(t_1, \ldots, t_n)$, we let $\pi(a) = p(\pi(t_1), \ldots, \pi(t_n))$ and similarly for a set of atoms. First-order semantic entailment is denoted by $\models$. It is well-known that, given two facts or CQs F and Q, $F \models Q$ iff there is a homomorphism from Q to F.

Definition 1 (Existential Rule)**.** *An* existential rule *(or simply* rule *when not ambiguous) is a first-order formula:*

$$R = \forall \mathbf{x} \forall \mathbf{y} \big(B[\mathbf{x}, \mathbf{y}] \rightarrow \exists \mathbf{z} H[\mathbf{y}, \mathbf{z}] \big),$$

where B is a conjunction, called the body *of R (also denoted by $body(R)$), and H is a conjunction called the* head *of R (denoted by $head(R)$). The* frontier *of R, denoted by $fr(R)$, is the set of variables $\mathbf{y} = vars(B) \cap vars(H)$ occurring both in the rule's body and head.*

Note that an existential rule could be equivalently defined as the formula $\forall \mathbf{y}(\exists \mathbf{x} B[\mathbf{x}, \mathbf{y}] \rightarrow \exists \mathbf{z} H[\mathbf{y}, \mathbf{z}])$. In the following, we will omit quantifiers since there is no ambiguity.

A *knowledge base* (KB) $\mathcal{K} = (F, \mathcal{R})$ is composed of a fact (in database terms: a database instance) F and a finite set of rules (in database terms: a TGD set) $\mathcal{R}$. W.l.o.g. we assume that the rules have pairwise *disjoint* sets of variables. We denote by $\mathcal{C}$ the set of constants occurring in $(F, \mathcal{R})$ and by T_0 (called the set of "initial terms") the set $vars(F) \cup \mathcal{C}$, i.e.,

5. Note that hereby we generalize the classical notion of a (ground) fact in order to take existential variables into account. This is in line with the notion of a *database instance* in database theory, where the existentially quantified variables are referred to as *labeled nulls*.

T_0 includes not only the terms from F but also the constants occurring in rules. Next, we formally define the problem considered by us.

Definition 2 (BCQ-Entailment)**.** *The decision problem of* entailment of Boolean conjunctive queries under existential rules *is defined as follows:*

- Name: BCQ-Entailment
- Input*: A knowledge base $\mathcal{K} = (F, \mathcal{R})$ and a Boolean conjunctive query Q.*
- Output*: YES iff $\mathcal{K} \models Q$, NO otherwise.*

Depending on which parts of the input are assumed to be fixed, we distinguish the following three complexity notions when investigating BCQ-Entailment:

- When considering *data complexity*, we assume $\mathcal{R}$ and Q to be fixed, only F (the data) can vary.
- When investigating *query complexity*, F and $\mathcal{R}$ are fixed and Q may vary.
- In case of *combined complexity*, F, $\mathcal{R}$ and Q can all change arbitrarily.

We now define the fundamental notions of rule application and $\mathcal{R}$-derivation, which we relate to the *chase* procedure in databases.

Definition 3 (Application of a Rule)**.** *A rule R is* applicable *to a fact F if there is a homomorphism π from $\mathrm{body}(R)$ to F; the result of the* application of R to F w.r.t. π *is a fact $\alpha(F, R, \pi) = F \cup \pi^{\mathrm{safe}}(\mathrm{head}(R))$ where π^{safe} is a substitution of $\mathrm{head}(R)$, which replaces each $x \in \mathrm{fr}(R)$ with $\pi(x)$, and each other variable with a fresh variable. As $\alpha(F, R, \pi)$ does not depend on the whole π, but only on $\pi_{|\mathrm{fr}(R)}$ (the restriction of π to $\mathrm{fr}(R)$), we also write $\alpha(F, R, \pi_{|\mathrm{fr}(R)})$.*

Example 3. *Let $F = \{r(a, b), r(c, d), p(d)\}$ and $R = r(x, y) \rightarrow r(y, z)$. There are two applications of R to F, respectively by $\pi_1 = \{x \mapsto a, y \mapsto b)\}$ and $\pi_2 = \{x \mapsto c, y \mapsto d\}$. We obtain $F_1 = \alpha(F, R, \pi_1) = F \cup \{r(b, z_1)\}$ and $F_2 = \alpha(F, R, \pi_2) = F \cup \{r(d, z_2)\}$.*

Definition 4 ($\mathcal{R}$-Derivation)**.** *Let F be a fact and $\mathcal{R}$ be a set of rules. An $\mathcal{R}$-derivation (from F to F_k) is a finite sequence $(F_0 = F), (R_0, \pi_0, F_1), \ldots, (R_{k-1}, \pi_{k-1}, F_k)$ such that for all $0 \leq i < k$, $R_i \in \mathcal{R}$ and π_i is a homomorphism from $\mathrm{body}(R_i)$ to F_i such that $F_{i+1} = \alpha(F_i, R_i, \pi_i)$. When only the successive facts are needed, we note $(F_0 = F), F_1, \ldots, F_k$.*

The following theorem essentially stems from earlier results on conceptual graph rules (Salvat & Mugnier, 1996).

Theorem 1 (Soundness and Completeness of $\mathcal{R}$-Derivations)**.** *Let $\mathcal{K} = (F, \mathcal{R})$ be a KB and Q be a CQ. Then $F, \mathcal{R} \models Q$ iff there exists an $\mathcal{R}$-derivation from F to some F_k such that $F_k \models Q$.*

It follows that a breadth-first forward chaining mechanism yields a positive answer in finite time when $\mathcal{K} \models Q$. Let $F_0 = F$ be the initial fact. Each step is as follows: (1) check if Q maps by homomorphism to the current fact, say F_{i-1} at step i ($i > 0$): if it is the case, Q has a positive answer; (2) otherwise, produce a fact F_i from F_{i-1}, by computing all new homomorphisms from each rule body to F_{i-1}, then performing all corresponding rule applications. A homomorphism to F_{i-1} is said to be *new* if it has not been already computed at a previous step, i.e., it uses at least an atom added at step $i-1$. The fact F_k obtained at the end of step k is called the *k-saturation* of F and is denoted by $\alpha_k(F, \mathcal{R})$; we define the *saturation* of F by $\mathcal{R}$ as $\alpha_\infty(F, \mathcal{R}) = \cup_{k \geq 0} \alpha_k(F, \mathcal{R})$.

Preceding notions are closely related to classical database notions. Forward chaining (with existential rules) is known as the *chase* (with TGDs) (Maier et al., 1979; Aho, Beeri, & Ullman, 1979). Hence, the notion of an $\mathcal{R}$-derivation corresponds to a chase sequence. The chase is seen as a tool for computing the saturation of a database with respect to a set of TGDs. Several variants of the chase are known, which all produce a result homomorphically equivalent to $\alpha_\infty(F, \mathcal{R})$. The chase yields a *canonical model* of $(F, \mathcal{R})$, which is isomorphic to the output of the chase, and has the property of being *universal*, which means that it maps by homomorphism to any model of $(F, \mathcal{R})$. It follows that $(F, \mathcal{R}) \models Q$ if and only if Q maps by homomorphism to $\alpha_\infty(F, \mathcal{R})$ (Deutsch, Nash, & Remmel, 2008) (and (Baget, Leclère, Mugnier, & Salvat, 2011) in the setting of existential rules).

We now formally specify some other notions that we have already introduced informally. A fact can naturally be seen as a hypergraph whose nodes are the terms in the fact and whose hyperedges encode the atoms. The *primal graph* of this hypergraph has the same set of nodes and there is an edge between two nodes if they belong to the same hyperedge. The *treewidth* of a fact is defined as the treewidth of its associated primal graph. Given a fact F_i, a derivation S yielding F_i, or a tree decomposition $\mathfrak{T}$ of F_i, we let $atoms(S) = atoms(\mathfrak{T}) = F_i$.

Definition 5 (Tree Decomposition and Treewidth of a Fact)**.** *Let F be a (possibly infinite) fact. A* tree decomposition *of F is a (possibly infinite) tree $\mathfrak{T}$, with set of nodes $\mathcal{B} = \{B_0, \ldots, B_k, \ldots\}$, and two functions $terms : \mathcal{B} \to 2^{terms(F)}$ and $atoms : \mathcal{B} \to 2^F$, where:*

1. *$\bigcup_i terms(B_i) = terms(F)$;*
2. *$\bigcup_i atoms(B_i) = F$;*
3. *For each $B_i \in \mathcal{B}$ holds $terms(atoms(B_i)) \subseteq terms(B_i)$;*
4. *For each term e in F, the subgraph of $\mathfrak{T}$ induced by the nodes B_i with $e \in terms(B_i)$ is connected ("Running intersection property").*

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

A set of rules $\mathcal{R}$ is called a *bounded-treewidth set (bts)* if for any fact F there exists an integer b such that the treewidth of any fact F' that can be $\mathcal{R}$-derived from F is less or equal to b. The ENTAILMENT problem is decidable when $\mathcal{R}$ is *bts* (Calì et al., 2008; Baget et al., 2011). The main argument of the proof, introduced by Calì et al., relies on the observation that $\mathcal{K} \wedge \neg Q$ enjoys the bounded-treewidth model property, i.e, has a model with bounded treewidth when it is satisfiable, i.e., when $\mathcal{K} \not\models Q$. Decidability follows from the decidability

of the satisfiability problem for classes of first-order formulas having the bounded-treewidth property, a result from Courcelle (Courcelle, 1990). However, the proof of this latter result does not lead (or at least not directly) to an algorithm for BCQ-Entailment under *bts* rules. We now focus on "concrete" subclasses of *bts*.

A rule R is *guarded* (*g*) if there is an atom $a \in body(R)$ with $vars(body(R)) \subseteq vars(a)$. We call a a guard of the rule. R is *weakly guarded* (*wg*) if there is $a \in body(R)$ (called a weak guard) that contains all affected variables from $body(R)$. The notion of affected variable is relative to the rule set: a variable is affected if it occurs only in affected predicate positions, which are positions that may contain an existential variable generated by forward chaining (Fagin et al., 2005). More precisely, the set of affected positions w.r.t. $\mathcal{R}$ is the smallest set that satisfies the following conditions: (1) if there is a rule head containing an atom with predicate p and an existentially quantified variable in position i, then position (p, i) is affected; (2) if a rule body contains a variable x appearing in affected positions only and x appears in the head of this rule in position (q, j) then (q, j) is affected. The important point is that a rule application necessarily maps non-affected variables to terms from the initial fact (and more generally to T_0 in the case where rules may add constants). The *g* and *wg* rule classes were described and their complexity was analyzed by Calì et al. (2008, 2013).

R is *frontier-one* (*fr1*) if $|fr(R)| = 1$ and it is *guarded frontier-one* (*gfr1*) if it is both *g* and *fr1*. R is *frontier-guarded* (*fg*) if there is $a \in body(R)$ with $vars(fr(R)) \subseteq vars(a)$. The weak versions of these classes—*weakly frontier-one* (*wfr1*), *weakly guarded frontier-one* (*wgfr1*) and *weakly frontier-guarded* (*wfg*) rules—are obtained by relaxing the above criteria so that they only need to be satisfied by the affected variables. The syntactic inclusions holding between these *bts* subclasses are displayed in Fig. 1.

3. Greedy Bounded-Treewidth Sets of Rules

This section introduces greedy bounded-treewidth sets of rules (*gbts*). It is pointed out that *gbts* strictly contains the *wfg* class. However, in some sense, *gbts* is not more expressive than *wfg*: indeed, we exhibit a polynomial translation τ from any KB $\mathcal{K} = (F, \mathcal{R})$ to another KB $\tau(\mathcal{K}) = (\tau(F), \tau(\mathcal{R}))$ with $\tau(\mathcal{R})$ being *wfg*, which satisfies the following property: if $\mathcal{R}$ is *gbts*, then $\mathcal{K}$ and $\tau(\mathcal{K})$ are equivalent. This translation can thus be seen as a polynomial reduction from BCQ-Entailment under *gbts* to BCQ-Entailment under *wfg*.

3.1 Definition of the *gbts* Class

In a greedy derivation, every rule application maps the frontier of the rule in a special way: all the frontier variables that are mapped to terms introduced by rule applications are jointly mapped to variables added by one *single* previous rule application.

Definition 6 (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 $j < i$ such that $\pi_i(fr(R_i)) \subseteq vars(A_j) \cup vars(F_0) \cup \mathcal{C}$, where $A_j = \pi_j^{\text{safe}}(head(R_j))$.*

Note that, in the above definition, any $j < i$ can be chosen if $fr(R_i)$ is mapped to $vars(F_0) \cup \mathcal{C}$.

Example 4 (Non-Greedy Derivation). *Let $\mathcal{R} = \{R_0, R_1\}$ where:*

$$\begin{array}{rcl} R_0 & = & r_1(x,y) \rightarrow r_2(y,z) \text{ and} \\ R_1 & = & r_1(x,y) \wedge r_2(x,z) \wedge r_2(y,t) \rightarrow r_2(z,t) \end{array}$$

Let $F_0 = \{r_1(a,b) \wedge r_1(b,c)\}$ and $S = F_0, \ldots, F_3$ with:

$$\begin{array}{rcll} F_1 & = & \alpha(F_0, R_0, \{(y \mapsto b)\}) & \text{with } A_0 = \{r_2(b,x_1)\}, \\ F_2 & = & \alpha(F_1, R_0, \{(y \mapsto c)\}) & \text{with } A_1 = \{r_2(c,x_2)\}, \\ F_3 & = & \alpha(F_2, R_1, \pi_2) & \text{with } \pi_{2|\mathrm{fr}(R_1)} = \{z \mapsto x_1, t \mapsto x_2\} \end{array}$$

Then $\mathrm{fr}(R_1) = \{z,t\}$ is mapped to newly introduced variables in F_3, however, there is no A_j such that $\{\pi_2(z), \pi_2(t)\} \subseteq \mathrm{vars}(A_j)$. Thus S is not greedy.

Any greedy derivation can be associated with a so-called derivation tree, formally defined below. Intuitively, the root of the tree corresponds to the initial fact F_0, and each other node corresponds to a rule application of the derivation. Each node is labeled by a set of terms and a set of atoms. The set of terms assigned to the root is T_0, i.e., it includes the constants that are mentioned in rule heads. Moreover, T_0 is included in the set of terms of all nodes. This ensures that the derivation tree is a decomposition tree of the associated derived fact.

Definition 7 (Derivation Tree). *Let $S = (F_0 = F), \ldots, F_k$ be a greedy derivation. The* derivation tree *assigned to S, denoted by $DT(S)$, is a tree $\mathfrak{T}$ with nodes $\mathcal{B} = \{B_0, \ldots, B_k, \ldots\}$ and two functions $\mathrm{terms} : \mathcal{B} \rightarrow 2^{\mathrm{terms}(F_k)}$ and $\mathrm{atoms} : \mathcal{B} \rightarrow 2^{F_k}$, defined as follows:*

1. *Let $T_0 = \mathrm{vars}(F) \cup \mathcal{C}$. The root of the tree is B_0 with $\mathrm{terms}(B_0) = T_0$ and $\mathrm{atoms}(B_0) = \mathrm{atoms}(F)$.*
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 $\mathrm{terms}(B_i) = \mathrm{vars}(A_{i-1}) \cup T_0$ and $\mathrm{atoms}(B_i) = \mathrm{atoms}(A_{i-1})$. The parent of B_i is the node B_j for which j is the smallest integer such that $\pi_{i-1}(\mathrm{fr}(R_{i-1})) \subseteq \mathrm{terms}(B_j)$.*

The nodes of $DT(S)$ are also called bags.

Example 5 (Example 3 contd.). *We consider $F = \{r(a,b), r(c,d), p(d)\}$ and $R = r(x,y) \rightarrow r(y,z)$. We build $DT(S)$ for $S = (F_0 = F), (R, \pi_1, F_1), (R, \pi_2, F_2)$ as depicted in Figure 2. Let B_0 be the root of $DT(S)$. (R, π_1) yields a bag B_1 child of B_0, with $\mathrm{atoms}(B_1) = \{r(b, z_1)\}$ and $\mathrm{terms}(B_1) = \{a, b, c, d, z_1\}$. (R, π_2) yields a bag B_2 with $\mathrm{atoms}(B_2) = \{r(d, z_2)\}$ and $\mathrm{terms}(B_2) = \{a, b, c, d, z_2\}$. $\mathrm{fr}(R_0) = \{y\}$ and $\pi_2(y) = d$, which is both in $\mathrm{terms}(B_0)$ and $\mathrm{terms}(B_1)$, B_2 is thus added as a child of the highest bag, i.e., B_0. R can be applied again, with homomorphisms $\pi_3 = \{x \mapsto b, y \mapsto z_1\}$ and $\pi_4 = \{x \mapsto d, y \mapsto z_2\}$, which leads to create two bags, B_3 and B_4, under B_1 and B_2 respectively. Clearly, applications of R can be repeated indefinitely.*

Note that, in the second point of the definition of a derivation tree, there is at least one j with $\pi_{i-1}(fr(R_{i-1})) \subseteq terms(B_j)$ because S is greedy. The following property is easily checked, noticing that T_0 occurs in each bag, which ensures that the running intersection property is satisfied.

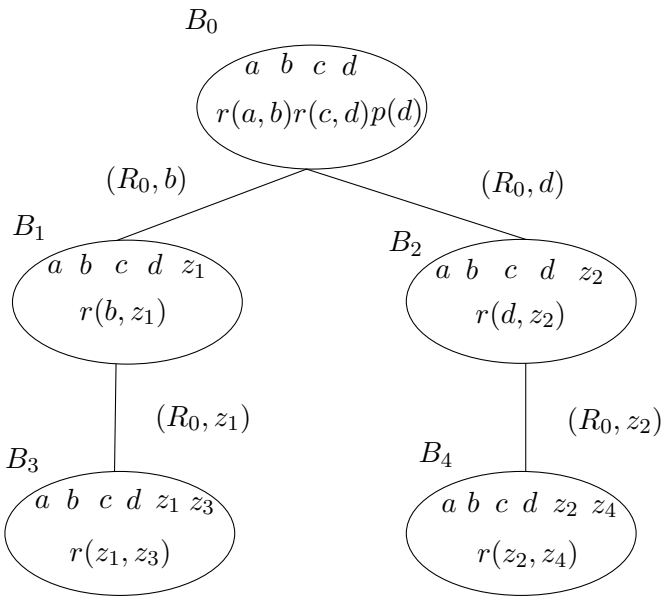

Figure 2: Derivation tree of Example 5. Only the image of the single frontier variable from R_0 is mentioned in edge labels.

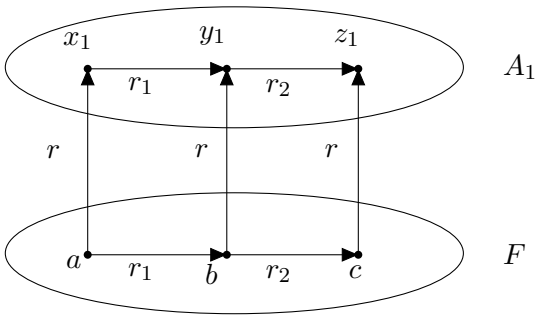

Figure 3: Illustration of Example 6

Property 2. *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 8 (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 class *gbts* is a strict subclass of *bts* and does not contain *fes* (e.g., in Example 4: $\mathcal{R}$ is *fes* but not *gbts*). It is nevertheless an expressive subclass of *bts* since it contains *wfg*:

Property 3. *Any set of wfg rules is 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. Let $a = \pi_i(g)$. Either $a \in F$ or $a \in A_j$ for some $j \leq i$. In the first case, $\pi_i(fr(R_i)) \subseteq terms(F) \subseteq vars(F_0) \cup \mathcal{C}$; in the second case, $\pi_i(fr(R_i)) \subseteq terms(A_j) \subseteq vars(A_j) \cup \mathcal{C}$. We conclude that $\mathcal{R}$ is *gbts*. □

The obtained inclusion is strict since there are *gbts* rule sets which are not *wfg* as shown in the following example.

Example 6 (*gbts* but not *wfg*)**.** *Let $R = r_1(x, y) \wedge r_2(y, z) \rightarrow r(x, x') \wedge r(y, y') \wedge r(z, z') \wedge r_1(x', y') \wedge r_2(y', z')$. $\{R\}$ 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 $F = \{r_1(a, b), r_1(b, c)\}$. R is applicable to F, which leads to create $r(a, x_1), r(b, y_1), r(c, z_1), r_1(x_1, y_1)$, and $r_2(y_1, z_1)$, as shown in Fig. 3. R is thus newly applicable, mapping its frontier to x_1, y_1, and z_1. This can be repeated infinitely often, therefore $\{R\}$ 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 of R to terms that have been created in the same bag (for instance, to the atoms in A_1 in Fig. 3), thus ensuring that $\{R\}$ is gbts.*

3.2 A Translation into Weakly Frontier-Guarded Rules

Next we will present a translation applicable to any set of existential rules. This translation can be computed in polynomial time, its result is always *wfg* and it is guaranteed to preserve query answers if the input is *gbts*.

The translation introduces two new predicates: a unary predicate *initial* and a predicate *samebag* of higher arity. Intuitively, *initial* will mark terms from the initial fact F, as well as constants added by rule applications, and *samebag* will gather terms that are "in the same bag".

Definition 9 (*wfg* Translation)**.** *Let $\mathcal{K} = (F, \mathcal{R})$ be a KB. The* wfg translation *of $\mathcal{K}$ is the KB $\tau(\mathcal{K}) = (\tau(F), \tau(\mathcal{R}))$ where $\tau(F) = F \cup \{initial(t) | t \in terms(F)\}$ and $\tau(\mathcal{R}) = \mathcal{R}^{\text{same}} \cup \mathcal{R}^{\text{trans}}$, where $\mathcal{R}^{\text{same}}$ and $\mathcal{R}^{\text{trans}}$ are defined as follows (where initial is a fresh unary predicate and* samebag *is fresh predicate with arity $q = \max(|terms(head(R))|_{R \in \mathcal{R}}) + |T_0|$):*

- *$\mathcal{R}^{\text{same}}$ contains the following rules:*

  $R_1^{\text{same}} = initial(x) \rightarrow samebag(x, \ldots, x)$,
  $R_2^{\text{same}} = samebag(x_1, x_2, \ldots, x_q) \wedge initial(x) \rightarrow samebag(x, x_2 \ldots, x_q)$,
  one rule of the following type for each $1 \leq i \leq q$:
  $R_{3i}^{\text{same}} = samebag(x_1, \ldots, x_i, \ldots, x_q) \rightarrow samebag(x_i, \ldots, x_1, \ldots x_q)$, *and*
  $R_4^{\text{same}} = samebag(x_1, \ldots, x_{q-1}, x_q) \rightarrow samebag(x_1, \ldots, x_{q-1}, x_1)$.

- *$\mathcal{R}^{\text{trans}}$ contains one translated rule $\tau(R)$ for every rule R from $\mathcal{R}$: for some rule $R = B[\mathbf{x}, \mathbf{y}] \rightarrow H[\mathbf{y}, \mathbf{z}]$ with $c_1, \ldots, c_k$ being the constants occurring in H, we let $\tau(R) = B[\mathbf{x}, \mathbf{y}] \wedge samebag(\mathbf{y}, \mathbf{v}) \rightarrow H[\mathbf{y}, \mathbf{z}] \wedge samebag(\mathbf{y}, \mathbf{z}, \mathbf{w}) \wedge_{i:1,\ldots,k} initial(c_i)$, where $\mathbf{w} \subseteq \mathbf{v}$ and $\mathbf{v}$ is a set of fresh variables.*

Intuitively, Rules R_1^{same} and R_2^{same} express that the initial terms (as well as constants added by rule applications) are in all bags; rules R_{3i}^{same} and rule R_4^{same} respectively allow any permutation and any duplication of arguments in an atom with predicate *samebag*. In the translation of the rules from $\mathcal{R}$, the sets of variables $\mathbf{v}$ and $\mathbf{w}$ are used to fill the atoms with predicate *samebag* to obtain arity q.

Property 4. *For any set $\mathcal{R}$ of existential rules, $\tau(\mathcal{R})$ is wfg.*

Proof. $\mathcal{R}^{\text{same}} \setminus \{R_2^{\text{same}}\}$ is *guarded*. R_2^{same} is *fg*. No rule affects the position in the unary predicate *initial*, thus all affected variables in R_2^{same} are guarded by the atom with predicate *samebag*, hence $\tau(\mathcal{R})$ is *wfg*. □

We next establish that, assuming we do not consider consequences involving *initial* or *samebag*, $\tau(\mathcal{R})$ is sound with respect to $\mathcal{R}$ and it is even complete in case $\mathcal{R}$ is *gbts*.

Property 5. *For any Boolean CQ Q over the initial vocabulary, 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 vocabulary).*

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 $\mathcal{R}^{\text{same}}$ and replacing each application of a rule $\tau(R_i)$ by an application of the rule R_i with ignoring the atoms with predicate *samebag*. Moreover, the facts respectively obtained by both derivations are equal on the initial vocabulary (i.e., when considering only the atoms with predicate in the initial vocabulary, 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) for all i such that $0 \leq i \leq k$, F_i and F'_i are equal on the initial vocabulary; and, (b) for all i such that $0 \leq i < k$, 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 $\ell = 0$. Assume it is true for $\ell = n$. Consider the application of R_n with homomorphism π_n from $body(R_n)$ to F_n. We note $fr(R_n) = \{y_1 \ldots y_p\}$ such that $body(\tau(R_n))$ contains the atom $samebag(y_1, \ldots, y_p, \ldots)$. Since $\mathcal{R}$ is *gbts*, there is an A_j such that some variables from $fr(R_n)$, say $y_{i_1} \ldots y_{i_m}$ are mapped to $vars(A_j)$, and the remaining variables from $fr(R_n)$, say $y_{i_{m+1}} \ldots y_{i_p}$ are mapped to $T_0 = vars(F) \cup \mathcal{C}$. The application of $\tau(R_j)$ in $\mathcal{S}'$ has produced a *samebag* atom s_1 that contains $\pi_n(y_{i_1}) \ldots \pi_n(y_{i_m})$ (by induction hypothesis (b)). By applying Rules R^{same}_{3i} and Rule R^{same}_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 $s_2 = samebag(\pi_n(y_{i_1}), \ldots, \pi_n(y_{i_m}), \ldots)$. Then, with Rule R^{same}_2, we add each $\pi_n(y_{i_j})$ for $m < j \leq p$ (note that F'_n necessarily contains $initial(\pi_n(y_{i_j}))$) and build the atom $s_3 = samebag(\pi_n(y_{i_{m+1}}), \ldots, \pi_n(y_{i_p}), \pi_n(y_{i_1}) \ldots \pi_n(y_{i_m}), \ldots)$. Finally, with Rules R^{same}_{3i}, we permute the p first arguments in s_3 to obtain $s_4 = samebag(\pi_n(y_1), \ldots, \pi_n(y_p), \ldots)$. Since F_n and F'_n are equal on the initial vocabulary by induction hypothesis (a), the fact obtained from F'_n after application of the previous rules from $\mathcal{R}^{\text{trans}}$ is still equal to F_n on the initial vocabulary. We build π'_n by extending π_n such that the atom with predicate *samebag* in $body(\tau(R_n))$ is mapped to s_4. Parts (a) and (b) of the induction property are thus satisfied for $\ell = n + 1$. □

4. An Algorithm for *gbts*: PatSat

We give here an informal high-level description of the PatSat algorithm (for pattern saturation). Due to the existentially quantified variables in rule heads, a forward chaining mechanism does not halt in general. However, as we have seen in the preceding section, for *gbts*, each sequence of rule applications gives rise to a so-called *derivation tree*, which is a decomposition tree of the derived fact; moreover, this tree can be built in a *greedy* way: each rule application produces a new tree node B (called a *bag*), which contains the atoms created by the rule application, such that the derived fact is the union of all bag atoms from this tree. The derived fact is potentially infinite, but thanks to its tree-like structure, the forward chaining process can be stopped after a finite number of rule applications as some periodic behavior will eventually occur.

The PatSat algorithm proceeds in two steps: first, it computes a finite tree, called a *(full) blocked tree*, which finitely represents all possible derivation trees; second, it evaluates a query against this blocked tree. Building a blocked tree relies on the following notions:

- *bag patterns*: Each bag B is associated with a *pattern* P, which stores all ways of mapping any (subset of any) rule body to the current fact (that is: the intermediate fact associated with the tree at the current stage of the construction process), while using some terms from $terms(B)$. It follows that a rule is applicable to the current fact if and only if one of the bag patterns contains a mapping of its entire rule body. Then, the forward chaining can be performed "on the bag-level" by forgetting about the underlying facts and considering solely the derivation tree decorated with patterns.

At each step, patterns are maintained and kept up-to-date by a propagation procedure based on a *join* operation between the patterns of adjacent bags.

- an *equivalence* relation on bags: Thanks to patterns, an equivalence relation can be defined on bags, so that two bags are equivalent if and only if the "same" derivation subtrees can be built under them. The algorithm develops (that is: adds children nodes to) only one node per equivalence class, while the other nodes of that class are *blocked* (note, however, that equivalence classes evolve during the computation, thus a blocked node can later become non-blocked, and vice-versa). This tree grows until no new rule application can be performed to non-blocked bags: the *full blocked tree* is then obtained.

- *creation* and *evolution rules*: The equivalence relation that we propose is however not directly computable: the "natural" way to compute would require 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, as well as 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 first way to perform query answering is then to consider the query as a rule with a head reduced to a nullary prediate. In that case, it is enough to check if one pattern contains the entire body of this added rule. If one do not want to consider the query as a rule, one has to be more cautious. Indeed, the evaluation of a Boolean conjunctive query against a blocked tree cannot be performed by a simple homomorphism test. Instead, we define the notion of an APT-mapping, which can be seen as a homomorphism to an "unfolding" or "development" of this blocked tree. As the length of the developed paths that is relevant for query answering is bounded with an exponent that depends only on the rule set (more precisely, the exponent is the maximal number of variables shared by the body and the head of a rule), checking if there is an APT-mapping from a conjunctive query to a blocked tree is time polynomial in data complexity and nondeterministically time polynomial in query complexity.

In order to illustrate the numerous definitions of this section, we will employ 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 expressible by means of description logics.

Example 7 (Running Example)**.** *Let us consider* $\mathcal{R}^{\text{ex}} = \{R_1^{\text{ex}}, \ldots, R_7^{\text{ex}}\}$ *defined as follows:*

- $R_1^{\text{ex}} = q_1(x_1, y_1, z_1) \rightarrow s(y_1, t_1) \wedge r(z_1, t_1) \wedge q_2(t_1, u_1, v_1)$*;*
- $R_2^{\text{ex}} = q_2(x_2, y_2, z_2) \rightarrow s(y_2, t_2) \wedge r(z_2, t_2) \wedge q_3(t_2, u_2, v_2)$*;*
- $R_3^{\text{ex}} = q_3(t_3, u_3, v_3) \rightarrow h(t_3)$*;*

- $R_4^{\text{ex}} = q_2(x_4, y_4, z_4) \wedge s(y_4, t_4) \wedge r(z_4, t_4) \wedge h(t_4) \rightarrow h(x_4) \wedge p_1(y_4) \wedge p_2(z_4)$;
- $R_5^{\text{ex}} = q_1(x_5, y_5, z_5) \wedge s(y_5, t_5) \wedge r(z_5, t_5) \wedge h(t_5) \rightarrow p_1(y_5) \wedge p_2(z_5)$;
- $R_6^{\text{ex}} = p_1(x_p) \wedge i(x_p) \rightarrow r(x_p, y_p) \wedge p_2(y_p) \wedge i(y_p)$;
- $R_7^{\text{ex}} = p_2(x_q) \wedge i(x_q) \rightarrow s(x_q, y_q) \wedge p_1(y_q) \wedge i(y_q)$.

The initial fact will be:

$$F^{\text{ex}} = q_1(a, b, c) \wedge q_1(d, c, e) \wedge q_1(f, g, g) \wedge i(c) \wedge i(g).$$

The subset $\{R_1^{\text{ex}}, R_2^{\text{ex}}, R_3^{\text{ex}}\}$ is a finite expansion set[6]. Applying these rules will create some existentially quantified variables. A first interesting phenomenon is that these existential variables allow to infer some new information about the initial terms. Last, R_4^{ex} and R_5^{ex} will generate infinitely many fresh existential variables, which will allow us to illustrate both the blocking procedure and the querying operation. While it can be argued that these rules are slightly complicated, it will allow to illustrate why we cannot block nodes without being careful.

Let us illustrate this rule set with an example of greedy derivation of F^{ex} under $\mathcal{R}^{\text{ex}}$.

Example 8. *Let us consider the following sequence of rule applications:*

- R_1^{ex} *is applied to* F^{ex} *by* $\pi_1 = \{x_1 \mapsto a, y_1 \mapsto b, z_1 \mapsto c\}$, *creating* $\{s(b, t_1^1), r(c, t_1^1), q_2(t_1^1, u_1^1, v_1^1)\}$.
- R_2^{ex} *is applied to* $\alpha(F^{\text{ex}}, R_1^{\text{ex}}, \pi_1)$ *by* $\pi_2 = \{x_2 \mapsto t_1^1, y_2 \mapsto u_1^1, z_2 \mapsto v_1^1\}$, *creating* $\{s(u_1^1, t_2^1)$, $r(v_1^1, t_2^1)$, $q_3(t_2^1, u_2^1, v_2^1)\}$
- R_3^{ex} *is applied on the resulting fact by* $\pi_3 = \{x_3 \mapsto t_1^2, y_3 \mapsto u_1^2, z_3 \mapsto v_1^2\}$, *creating a single new atom* $h(t_2^1)$.

This derivation is greedy, and its derivation tree is represented in Figure 4.

More generally, let us take a look at k-saturations of F^{ex} with respect to $\mathcal{R}^{\text{ex}}$. On F^{ex}, only R_1^{ex} is applicable by three 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 $\{s(g, t_1^3), r(g, t_1^3), q_2(t_1^3, u_1^3, v_1^3)\}$. $\alpha_1(F^{\text{ex}}, \mathcal{R}^{\text{ex}})$ is equal to the union of F^{ex} and these three sets of atoms. On $\alpha_1(F^{\text{ex}}, \mathcal{R}^{\text{ex}})$, three new rule applications are possible, each of them mapping the body of R_2^{ex} to one of the atoms with 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 $\alpha_2(F^{\text{ex}}, \mathcal{R}^{\text{ex}})$. On this fact, three new rule applications of R_3^{ex} 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^{ex}, 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^{ex} 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 $\alpha_5(F^{\text{ex}}, \mathcal{R}^{\text{ex}})$. R_6^{ex} and R_7^{ex} are now applicable, both mapping their frontier to c and g. They will create infinite branches.

6. Because, for example, their graph of rule dependency is acyclic (Baget et al., 2009)

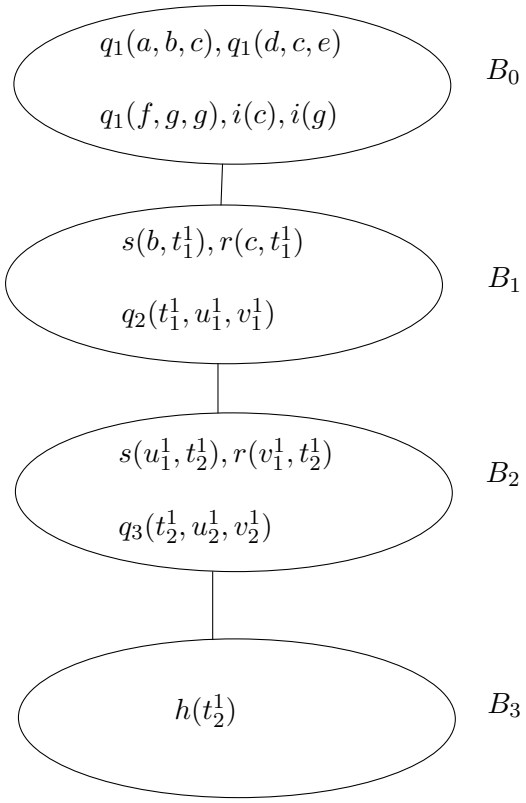

Figure 4: The derivation tree associated with Example 8

4.1 Patterned Forward Chaining

This section focuses on *bag patterns*. For all following considerations, we assume an arbitrary but fixed rule set $\mathcal{R}$ which is *gbts*. 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 formally capture some notion of "regularity" in a derivation tree, which will be exploited in the next section to finitely represent potentially infinite derivation trees.

Definition 10 (Pattern, Patterned Derivation Tree)**.** *A* pattern *of a bag B is a set of pairs (G, π), where G is a conjunction of atoms and π is a partial mapping from terms(G) to terms(B). G and π are possibly empty.*

For any $\mathcal{R}$-derivation S with derivation tree $DT(S)$, we obtain a patterned derivation tree*, noted $(DT(S), P)$, where P is a function assigning a pattern $P(B)$ to each bag B of $DT(S)$.*

The patterns that we consider are subsets of the rule bodies in $\mathcal{R}$.

Definition 11 (Pattern Soundness and Completeness)**.** *Let F_k be a fact obtained via a derivation S and let B be a bag in $(DT(S), P)$. $P(B)$ 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 the terms of B, i.e., $\pi' = \pi_{|\pi^{-1}(terms(B))}$. Finally, $(DT(S), P)$ 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 $(DT(S), P)$ is sound and complete w.r.t. F_k, a rule R is applicable to F_k iff there is a bag in $(DT(S), P)$ whose pattern contains a pair $(body(R), -)$; then, the bag created by a rule application (R, π) to F_k has parent B_j in $DT(S)$ iff B_j is the bag in $(DT(S), P)$ with the smallest j such that $P(B_j)$ contains $(body(R), \pi')$ for some π' which coincides with π on $fr(R)$, i.e., $\pi_{|fr(R)} = \pi'_{|fr(R)}$. Patterns are managed as follows: (1) The pattern of B_0 is the maximal 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 their 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 can be shown to be isomorphic to the derivation tree associated with the (regular) derivation.

Remember that our final rationale is to avoid computations on the "fact level". We will instead incrementally maintain sound and complete patterns by a propagation mechanism on patterns. This is why 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 12 (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 if 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 9. *Let us consider the bags B_1 and B_2 in Figure 4. Let $e_1 = (\{q_2(x_4, y_4, z_4)\}, \pi = \{x_4 \mapsto t_1^1, y_4 \mapsto u_1^1, z_4 \mapsto v_1^1\})$ be in the pattern of B_1, and $e_2 = (\{s(y_4, t_4), r(z_4, t_4), h(t_4)\}, \pi' = \{y_4 \mapsto u_1^1, z_4 \mapsto v_1^1, t_4 \mapsto t_2^1\})$ be in the pattern of B_2. 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 13 (Join)**.** *Let B_1 and B_2 be two bags with respective patterns $P(B_1) = P_1$ and $P(B_2) = P_2$. The join of P_1 with P_2, denoted $J(P_1, P_2)$, is the set of all defined $J(e_1, e_2)$, where $e_1 = (sb_R^1, \pi_1) \in P_1$, $e_2 = (sb_R^2, \pi_2) \in P_2$.*

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). Note that join preserves soundness, as stated in the following property.

Property 6. *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 the step from F_{i-1} to F_i in a (patterned) derivation sequence: let B_c be the bag created in this step and let B_p be its parent in $(DT(S), P)$.

Definition 14 (Initial Pattern)**.** *The* initial pattern *of a bag B_c, denoted by $P_{\text{init}}(B_c)$, is the set of pairs (G, π) such that G is a subset of some rule body of $\mathcal{R}$ and π is a* homomorphism *from G to atoms(B_c).*

Example 10 (Initial Pattern)**.** *Let us consider the initial pattern of B_2 in Figure 4. The atoms of B_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 $R_1^{\text{ex}}, R_2^{\text{ex}}, R_6^{\text{ex}}$ and R_7^{ex}, no subset of a rule body maps to the atoms of B_2. Thus, they do not contribute to the initial pattern of B_2. There is one homomorphism from the body of R_3^{ex} to atoms of B_2, and thus its initial pattern contains:

$$(\{q_3(t_3, u_3, v_3)\}, \{t_3 \mapsto t_2^1, u_3 \mapsto u_2^1, v_3 \mapsto v_2^1).$$

As for subsets of the body of R_4^{ex}, there are three elements added to the initial pattern of B_2:

- $(\{s(y_4, t_4)\}, \{y_4 \mapsto u_1^1, t_4 \mapsto t_2^1\})$,
- $(\{r(z_4, t_4)\}, \{t_4 \mapsto t_2^1, z_4 \mapsto v_1^1\})$,
- $(\{s(y_4, t_4), r(z_4, t_4)\}, \{y_4 \mapsto u_1^1, t_4 \mapsto t_2^1, z_4 \mapsto v_1^1\})$.

Similar elements are added by taking subsets of the body of R_5^{ex}.

Property 7 (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 $(G, \pi) \in P_{\text{init}}(B_c)$, π is a homomorphism from G to $atoms(B_c) \subseteq F_i$. □

Obviously, if a pattern is sound w.r.t. F_{i-1} then it is sound w.r.t. F_i. The following property focus on completeness.

Property 8 (Completeness of $J(P(B_c), P(B_p))$ w.r.t. F_i)**.** *Assume that* $P(B_p)$ *is complete w.r.t.* F_{i-1} *and* $\mathcal{R}$*. Then* $J(P_{init}(B_c), P(B_p))$ *is complete w.r.t.* F_i.

Proof. Let π be a homomorphism from $sb_R \subseteq body(R)$ to F_i, for some rule R. We show that $J(P_{\text{init}}(B_c), P(B_p))$ contains (sb_R, π'), where π' is the restriction of π to the inverse image of $terms(B_c)$. 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., $atoms(B_c)$. If b_i is not empty, by definition of the initial pattern, $P_{\text{init}}(B_c)$ contains (b_i, π_c), where π_c is the restriction of π to $terms(b_i)$. 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 $terms(B_p)$. If b_{i-1} or b_i is empty, (sb_R, π') belongs to $J(P_{\text{init}}(B_c), P(B_p))$. Otherwise, consider $J((b_i, \pi_c), (b_{i-1}, \pi_p))$: it is equal to (sb_R, π'). □

Property 9 (Completeness of Join-Based Propagation)**.** *Assume that* $(DT(S), P)$ *is complete w.r.t.* F_{i-1}*, and* $P(B_c)$ *is computed by* $J(P_{\text{init}}(B_c), P(B_p))$*. Let* $d(B)$ *denote the distance of a bag* B *to* B_c *in* $(DT(S), P)$*. 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* $(DT(S), P')$ *be obtained from* $(DT(S), P)$ *by updating all patterns by increasing value of* d *of the corresponding bags. Then* $(DT(S), P')$ *is complete w.r.t.* F_i.

Proof. From Property 8, we know that $P'(B_c)$ is complete w.r.t. F_i. It remains to prove the following property: let $P'(B)$ be obtained by computing $J(P(B), P'(B'))$; 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 8. If one of the subsets is empty, we are done. Otherwise, the partition allows 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 $e \in terms(B_c)$, hence, thanks to the running intersection property of a decomposition tree, $e \in terms(B')$, 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 derivation tree throughout the derivation:

1. initialize: compute $P_{\text{init}}(B_c)$ for the newly created pattern B_c;
2. update: $P'(B_c) = J(P_{\text{init}}(B_c), P(B_p))$;
3. propagate: first, propagate from $P(B_c)$ to $P(B_p)$, i.e., $P'(B_p) = J(P(B_p), P'(B_c))$; then, for each bag B updated from a bag B', update its children B_i (for $B_i \neq B'$) by $P'(B_i) = J(P(B_i), P'(B))$ and its parent B_j by $P'(B_j) = J(P(B_j), P'(B))$. Iterate this step until every pattern is updated (i.e., $P'(B)$ is determined for every bag B of the current derivation tree).

4.2 Bag Equivalence

We now show how bag patterns allow us to identify a certain kind of regularity in a derivation tree. We first need some technical, but nonetheless natural definitions. We start with the notion of a *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 15 (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 $\text{fr}(R)$ by $T_0 \cup V$. The* fusion *of $\text{fr}(R)$ induced by π, denoted by σ_π, is the substitution of $\text{fr}(R)$ by $\text{fr}(R) \cup T_0$ such that for every variable $x \in \text{fr}(R)$, if $\pi(x) \in V$ then $\sigma_\pi(x)$ is the smallest[7] variable y of $\text{fr}(R)$ such that $\pi(x) = \pi(y)$; otherwise $\sigma_\pi(x) = \pi(x) \in T_0$.*

Example 11. *Let us consider $R_2^{\text{ex}} = q_2(x_2, y_2, z_2) \rightarrow s(y_2, t_2) \wedge r(z_2, t_2) \wedge q_3(t_2, u_2, v_2)$. Let $\pi_1 = \{y_2 \mapsto y_0, z_2 \mapsto y_0\}$. The substitution of the frontier of R_2 induced by π_1 is defined by $\sigma_{\pi_1} = \{y_2 \mapsto y_2, z_2 \mapsto y_2\}$. Let b be a constant, and π_2 be a substitution of the frontier of R_1 defined by $\pi_2 = \{y_2 \mapsto b, z_2 \mapsto b\}$. The fusion of the frontier induced by π_2 is defined by $\sigma_{\pi_2} = \{y_2 \mapsto b, z_2 \mapsto b\}$. Last, if π_3 maps y_2 and z_2 to two different existentially quantified 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 equivalence relation on the bags of a derivation tree.

Definition 16 (Structural Equivalence)**.** *Let B and B' be two bags created by applications (R, π_i) and (R, π_j), respectively, of the same rule R. B and B' are* structurally equivalent*, written $B \simeq B'$ if the fusions of $\text{fr}(R)$ induced by the restrictions of π_i and π_j to $\text{fr}(R)$ are equal.*

We will see later that structural equivalence is not sufficient to formalize regularity in a derivation tree. However, there is already a strong similarity between structurally equivalent bags: the purpose of Definition 17 is to formalize it.

Definition 17 (Natural Bijection)**.** *Let B and B' be two structurally equivalent bags created by applications (R, π_i) and (R, π_j). 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 $x \in T_0$, let $\psi_{B \to B'}(x) = x$*

7. We assume variables to be totally ordered (for instance by lexicographic order).

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

The natural bijection is thus an isomorphism between two bags. 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 18 (Pattern Inclusion, Pattern Equivalence)**.** *Let* B *and* B' *be two bags, with respective patterns* $P(B)$ *and* $P(B')$. *We say that* $P(B')$ includes $P(B)$, *denoted by* $P(B) \sqsubseteq P(B')$, *if :*

- B *and* B' *are structurally equivalent, i.e.,* $B \simeq B'$,
- $P(B')$ *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 10 helps to understand why Definition 18 provides us with a good notion of pattern equivalence, by linking the equivalence of patterns to the applicability of rules on bags having these patterns. Let us note that this property does not hold if we put structural equivalence in place of pattern equivalence.

Property 10. *Let* S *be a derivation, and* B *and* B' *two bags of* $(DT(S), P)$ *such that* $P(B) \sim P(B')$. *If a rule* R *is applicable to* B *by* π, *then* R *is applicable to* B' *by* $\psi_{B \to B'} \circ \pi$.

Proof. Since R is applicable to B by π, $(body(R), \pi_{|fr(R)})$ belongs to $P(B)$. By definition of the equivalence of patterns, $(body(R), \psi_{B \to B'} \circ \pi_{|fr(R)})$ belongs to $P(B')$, which implies that R is applicable to B'. □

We now present how this equivalence relation will be used to finitely represent the (potentially infinite) set of derived facts. Intuitively, a blocked tree $\mathfrak{T}_b$ is a subtree (with the same root) of a patterned derivation tree $(DT(S), P)$ of a sufficiently large derivation sequence S. Additionally every bag in $\mathfrak{T}_b$ is marked by either "blocked" or "non-blocked". Assuming that we know which length of derivation is enough, $\mathfrak{T}_b$ is constructed such that it has the following properties:

- for each equivalence class appearing in $(DT(S), P)$, there is exactly one non-blocked node of $\mathfrak{T}_b$ of that class;
- if a bag B is blocked in $\mathfrak{T}_b$, it is a leaf, i.e., it has no child in $\mathfrak{T}_b$ (although it may have children in $(DT(S), P)$);
- if a bag is non-blocked in $\mathfrak{T}_b$, all children of B in $(DT(S), P)$ are present in $\mathfrak{T}_b$.

Definition 19 (Blocked Tree). *A blocked tree is a structure $(\mathfrak{T}_b, \sim)$, where $\mathfrak{T}_b$ is an initial segment of a patterned derivation tree and $\sim$ is the equivalence relation on the bags of $\mathfrak{T}_b$ such that for each $\sim$-class, all but one bag are said to be* blocked*; this non-blocked bag is called the* representative *of its class and is the only one that may have children.*

A blocked tree $\mathfrak{T}_b$ can be associated with a possibly infinite set of decomposition trees obtained by iteratively copying its bags. We first define the bag copy operation:

Definition 20 (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 $\psi_{B_1 \to B_2}$) is performed by adding a child B'_2 to B_2, such that $terms(B'_2) = \{\psi_{B'_1 \to B'_2}(t) \mid t \in terms(B'_1)\}$ and $atoms(B'_2) = \{\psi_{B'_1 \to B'_2}(a) \mid a \in atoms(B'_1)\}$, where $\psi_{B'_1 \to B'_2}$ is defined as follows: for all $x \in terms(B'_1)$, if $x \in terms(B_1)$ then $\psi_{B'_1 \to B'_2}(x) = \psi_{B_1 \to B_2}(x)$, otherwise $\psi_{B'_1 \to B'_2}(x)$ 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, which justifies the use of $\psi_{B'_1 \to B'_2}$ for the bijection.

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

Definition 21 (Trees Generated by a Blocked Tree). *Given a blocked tree $\mathfrak{T}_b$, let the* set $G(\mathfrak{T}_b)$ of trees generated by $\mathfrak{T}_b$ *be inductively defined as follows:*

- *Let B_0 be the root of $\mathfrak{T}_b$; the pair $(\{B_0\}, \{B_0 \mapsto B_0\})$ belongs to $G(\mathfrak{T}_b)$.*
- *Given a pair $(\mathfrak{T}, f) \in G(\mathfrak{T}_b)$, let B be a bag in $\mathfrak{T}$, and $B' = f(B)$; let B'_r be the representative of the $\sim$-class containing B' (i.e., $B'_r \neq B'$ if B' is blocked) and let B'_c be a child of B'_r. If B has no child mapped to B'_c by f, let $\mathfrak{T}_{new}$ be obtained from $\mathfrak{T}$ by copying B'_c under B (according to $\psi_{B'_r \to B}$), which yields a new bag B_c. Then $(\mathfrak{T}_{new}, f \cup (B_c \mapsto B'_c))$ belongs to $G(\mathfrak{T}_b)$.*

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

Note that a patterned decomposition tree thus generated is not necessarily a derivation tree, but it is an initial segment of a derivation tree. Among blocked trees, so-called *full blocked trees* are of particular interest.

Definition 22 (Full Blocked Tree). *A* full blocked tree $\mathfrak{T}^*$ *(of F and $\mathcal{R}$) is a blocked tree satisfying the two following properties:*

- *(Soundness) If $\mathfrak{T}'$ is generated by $\mathfrak{T}^*$, then there is some $\mathfrak{T}''$ generated by $\mathfrak{T}^*$ and an $\mathcal{R}$-derivation S from F such that $atoms(\mathfrak{T}'') = atoms(DT(S))$ (up to fresh variable renaming) and $\mathfrak{T}'$ is an initial segment of $\mathfrak{T}''$.*

- *(Completeness) For all $\mathcal{R}$-derivations from F, $DT(S)$ is generated by $\mathfrak{T}^*$.*

The procedure outlined above (considering a particular tree prefix of a sufficiently large derivation tree) is however not constructive. We show how to circumvent this problem in the next section.

4.3 Abstract Patterns and Abstract Pattern Saturation

We now aim at computing a full blocked tree. To this end, 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 on an abstract level 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 about the derivation trees that can be created starting from a fact and a set of rules: creation rules and evolution rules will be defined. In the 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. In order to have a uniform presentation, we consider the initial fact as a rule with empty body.

Definition 23 (Abstract Bag, Frontier Terms, Generated Variables)**.** *Let R be a rule from $\mathcal{R}$ and σ a fusion of $\mathrm{fr}(R)$. The* abstract bag *associated with R and σ (notation: $\mathbb{B}(R,\sigma)$) is defined by $\mathit{terms}(\mathbb{B}(R,\sigma)) = \sigma(\mathit{terms}(\mathit{head}(R))) \cup T_0$ and $\mathit{atoms}(\mathbb{B}(R,\sigma)) = \sigma(\mathit{head}(R))$. The* frontier terms *of $\mathbb{B}(R,\sigma)$ are the elements of $\sigma(\mathrm{fr}(R))$. Variables from $\mathit{terms}(\mathbb{B}(R,\sigma))$ that are not frontier terms are called* generated variables*.*

The notion of the natural bijection between structurally equivalent bags is extended to abstract bags in the straightforward way (note that there is exactly one abstract bag per structural equivalence class).

Example 12 (Abstract Bag)**.** *Let us consider $R_2^{\mathrm{ex}} = q_2(x_2,y_2,z_2) \to s(y_2,t_2) \wedge r(z_2,t_2) \wedge q_3(t_2,u_2,v_2)$, and three fusions of its frontier, namely: $\sigma_{\pi_1} = \{y_2 \mapsto y_2, z_2 \mapsto y_2\}$, $\sigma_{\pi_2} = \{y_2 \mapsto b, z_2 \mapsto b\}$ and $\sigma_{\pi_3} = \{y_2 \mapsto y_2, z_2 \mapsto z_2\}$. The abstract bag $\mathbb{B}(R_2^{\mathrm{ex}}, \sigma_{\pi_1})$ associated with R_2^{ex} and 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 terms of the abstract bag $\mathbb{B}(R_2^{\mathrm{ex}}, \sigma_{\pi_2})$ are $\{b,t_2,u_2,v_2\}$; its atoms are $\{s(b,t_2), r(b,t_2), q_3(t_2,u_2,v_2)\}$. For $\mathbb{B}(R_2^{\mathrm{ex}}, \sigma_{\pi_3})$, its terms are $\{y_2,t_2,u_2,v_2,z_2\}$ and its atoms are $\{s(y_2,t_2), r(z_2,t_2), q_3(t_2,u_2,v_2)\}$.*

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. To distinguish the abstract bags and patterns from their concrete counterparts, we will denote them by $\mathbb{B}$ and $\mathbb{P}$ (possibly with subscripts) instead of B and P.

Definition 24 (Abstract Pattern, Support)**.** *Let $\mathcal{R}$ be a set of rules, R be a rule and σ be a fusion of $\mathrm{fr}(R)$. An* abstract pattern $\mathbb{P}$ *with* support $\mathbb{B} = \mathbb{B}(R,\sigma)$ *is a set of pairs (G,π) where G is a subset of a rule body (of some rule of $\mathcal{R}$) and π is a partial mapping from $\mathit{terms}(G)$ to $\mathit{terms}(\mathbb{B})$. G and π are possibly empty.*

Example 13 (Abstract Pattern)**.** *Let us consider again the initial pattern described in Example 10. This pattern contains the following elements:*

- $(\{q_3(t_3, u_3, v_3)\}, \{t_3 \mapsto t_2^1, u_3 \mapsto u_2^1, v_3 \mapsto v_2^1\})$,
- $(\{s(y_4, t_4)\}, \{t_4 \mapsto t_2^1, y_5 \mapsto u_1^1\})$,
- $(\{r(z_4, t_4)\}, \{t_4 \mapsto t_2^1, z_4 \mapsto v_1^1\})$,
- $(\{s(y_4, t_4), r(z_4, t_4)\}, \{t_4 \mapsto t_2^1, y_4 \mapsto u_1^1, t_4 \mapsto v_1^1\})$,
- $(\{s(y_5, t_5)\}, \{t_5 \mapsto t_2^1, y_5 \mapsto u_1^1\})$,
- $(\{r(z_5, t_5)\}, \{t_5 \mapsto t_2^1, z_5 \mapsto v_1^1\})$,
- $(\{s(y_5, t_5), r(z_5, t_5)\}, \{t_5 \mapsto t_2^1, y_5 \mapsto u_1^1, z_5 \mapsto v_1^1\})$.

This pattern is associated with a bag equivalent to the abstract bag $\mathbb{B}(R_2, id)$. Thus, the abstract pattern $\mathbb{P}$ associated with this initial pattern contains the same elements, where the 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 25 (Initial Abstract Pattern)**.** *Let $\mathbb{B}$ be an abstract bag. The* initial abstract pattern *of $\mathbb{B}$, denoted by $\mathbb{P}_{\text{init}}(\mathbb{B})$ is the set of pairs (G, π) such that G is a subset of a rule body and π is a (full) homomorphism from G to atoms$(\mathbb{B})$.*

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. Let us assume that B_1 is structurally equivalent to an abstract bag $\mathbb{B}_1$ and that B_2 is structurally equivalent to an abstract bag $\mathbb{B}_2$. If we only state that a bag equivalent to $\mathbb{B}_2$ is a child of a bag equivalent to $\mathbb{B}_1$, we miss some information about the above mentioned shared terms. Capturing this information is the purpose of the notion of *link*.

Definition 26 (Link)**.** *Let $\mathbb{B}_1$ and $\mathbb{B}_2$ be two abstract bags. A* link *from $\mathbb{B}_2$ to $\mathbb{B}_1$ is an injective mapping λ from the frontier terms of $\mathbb{B}_2$ to the terms of $\mathbb{B}_1$ such that the range of λ has a non-empty intersection with the generated terms of $\mathbb{B}_1$.*

Please note that we define a link from a bag to its parent. It ensures that each bag has exactly one link. We will thus refer without ambiguity to the link of an abstract bag.

Example 14 (Link)**.** *Let us consider $R_1^{\text{ex}} = q_1(x_1, y_1, z_1) \rightarrow s(y_1, t_1) \wedge r(z_1, t_1) \wedge q_2(t_1, u_1, v_1)$ and $R_2^{\text{ex}} = q_2(x_2, y_2, z_2) \rightarrow s(y_2, t_2) \wedge r(z_2, t_2) \wedge q_3(t_2, u_2, v_2)$, and the two abstract bags $\mathbb{B}_1 = \mathbb{B}(R_1, id)$ and $\mathbb{B}_2 = \mathbb{B}(R_2, id)$. Then $\lambda = \{y_2 \mapsto u_1, z_2 \mapsto v_1\}$ is a link from $\mathbb{B}_2$ to $\mathbb{B}_1$.*

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

Definition 27 (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 $\mathbb{B}_1$ and $\mathbb{B}_2$ be two abstract bags such that $\mathbb{B}_1 \simeq B_1$ and $\mathbb{B}_2 \simeq B_2$. The link* induced *by B_1 and B_2 is the mapping λ of the frontier terms of $\mathbb{B}_2$ to terms$(\mathbb{B}_1)$ defined by $\lambda(y) = \psi_{B_1 \rightarrow \mathbb{B}_1}(\psi_{\mathbb{B}_2 \rightarrow B_2}(y))$. We then also say that B_2 is linked to B_1 by λ.*

Previous Property 10 states that the pattern of a bag determines the rules that are applicable on it. We will thus gather information relative to the structure of derivation trees by means of "saturation rules" whose intuition is explained by the following example. Note that these rules have nothing to do with existential rules.

Example 15. *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_1 *be the following pattern:* $\{r(x, y), \{x \mapsto a, y \mapsto b\}\}$. *For any bag* B *of a derivation tree* $DT(S)$ *such that* $P_1 \sqsubseteq P(B)$. R_1 *is applicable by mapping* x_1 *to* a *and* y_1 *to* b. *This allows to derive* $s(a, b)$ *(which may be used to apply* R_2*). Thus, the pattern of* B *in some derivation starting with* S *contains* $P_2 = \{(r(x_1, y_1), \{x_1 \mapsto a, y_1 \mapsto b\}), (s(x_2, y_2), \{x_2 \mapsto a, y_2 \mapsto b\})\}$. *Let us point out that this pattern inclusion is valid in "sufficiently complete" derivations, but not necessarily in the derivation tree of each derivation sequence.*

Example 15 gives the intuition behind *evolution rules*: it exhibits a case where we can infer that if a bag has a pattern including P_1, then its pattern can evolve into a pattern including P_2. Such information will be gathered by evolution rules, and will be denoted by $\mathbb{P}_1 \rightsquigarrow \mathbb{P}_2$ with $\mathbb{P}_1$ and $\mathbb{P}_2$ being the abstract counterparts of P_1 and P_2, respectively. To deal with the creation of new bags, we design *creation rules*. They allow us to derive information about the children that a bag with a given pattern must have. Such a rule will be denoted by $\mathbb{P}_1 \rightsquigarrow \lambda.\mathbb{P}_2$, and intuitively means that rules may be applied to ensure that any bag B_1 with pattern P_1 has a child B_2 with pattern P_2 such that the link induced by B_1 and B_2 is λ and $\mathbb{P}_1$ and $\mathbb{P}_2$ are again the abstract counterparts of P_1 and P_2, respectively.

In the following, we show how to derive a set of *sound* creation and evolution rules by means of Properties 11 to 16.

Definition 28 (Sound Creation Rule)**.** *Let* $\mathbb{P}_1, \mathbb{P}_2$ *be two abstract patterns, and* λ *be a link between the support of* $\mathbb{P}_2$ *and the support of* $\mathbb{P}_1$. *A* creation rule *is a rule of the following form:*

$$\gamma_c : \mathbb{P}_1 \rightsquigarrow \lambda.\mathbb{P}_2.$$

γ_c *is* sound *if for any derivation* S*, for any bag* B_1 *of* $(DT(S), P)$ *such that* $\mathbb{P}_1 \sqsubseteq P(B_1)$*, there exists a derivation* S' *extending* S *with a child* B_2 *of* B_1 *in* $(DT(S'), P')$ *linked by* λ *to* B_1*, and for which* $\mathbb{P}_2 \sqsubseteq P'(B_2)$.

Definition 29 (Sound Evolution Rule)**.** *Let* $\mathbb{P}_1, \mathbb{P}_2$ *be two abstract patterns. An* evolution rule *is a rule of the following form:*

$$\gamma_e : \mathbb{P}_1 \rightsquigarrow \mathbb{P}_2.$$

γ_e *is* sound *if* $\mathbb{P}_1 \sqsubseteq \mathbb{P}_2$ *and for any derivation* S *and for any bag* B *of* $(DT(S), P)$ *satisfying* $\mathbb{P}_1 \sqsubseteq P(B)$*, there exists a derivation* S' *extending* S *with patterned derivation tree* $(DT(S'), P')$ *such that* $\mathbb{P}_2 \sqsubseteq P'(B)$.

We now exhibit properties allowing to build sound rules.

Property 11. *Let* $\mathbb{P}$ *be an abstract pattern with support* $\mathbb{B}$*, let* R *be a rule from* $\mathcal{R}$*, and let* π *be a mapping from* $fr(R)$ *to* terms$(\mathbb{B})$ *such that its range has a non empty intersection*

with the generated terms in $\mathbb{P}$. *Let* $(body(R), \pi)$ *be an element of* $\mathbb{P}$. *Let* σ *be the fusion of* $fr(R)$ *induced by* π. *Then* $\mathbb{P} \rightsquigarrow \lambda.\mathbb{P}_{\text{init}}(\mathbb{B}(R, \sigma))$ *is a sound creation rule, where* λ *is equal to* π *restricted to* $\{\sigma(y) \mid y \in fr(R)\}$.

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

We now define the counterpart of elementary joins for abstract patterns. The main difference is that the relationships between terms of different abstract patterns cannot be checked by equality as it was done previously. 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 30 (Elementary Abstract Upper/Lower Join)**.** *Let* $\mathbb{P}_1$ *and* $\mathbb{P}_2$ *be two abstract patterns, and let* λ *be a link from* $\mathbb{P}_2$ *to* $\mathbb{P}_1$. *Let* R *be a rule in* $\mathcal{R}$ *and let* $(sb_1, \pi_1) \in \mathbb{P}_1$ *and* $(sb_2, \pi_2) \in \mathbb{P}_2$ *for* $sb_1, sb_2 \subseteq body(R)$. *The* elementary abstract upper and lower joins *of* (sb_1, π_1) *with* (sb_2, π_2) *are defined if* $\pi_1(x)$ *and* $\lambda(\pi_2(x))$ *are defined and equal for all* $x \in vars(sb_1) \cap vars(sb_2)$. *In that case, it is the pair* $(sb_1 \cup sb_2, \pi)$ *with:*

- $\pi = \pi_1 \cup \lambda \circ \pi_2'$, *where* π_2' *is the restriction of* π_2 *to* $\pi_2^{-1}(domain(\lambda))$, *for the upper join;*
- $\pi = \pi_2 \cup \lambda^{-1} \circ \pi_1'$, *where* π_1' *is the restriction of* π_1 *to* $\pi_1^{-1}(\text{range}(\lambda))$, *for the lower join.*

Definition 31 (Abstract Upper/Lower Join)**.** *Let* $\mathbb{P}_1$ *and* $\mathbb{P}_2$ *be two abstract patterns, and let* λ *be a link from* $\mathbb{P}_2$ *to* $\mathbb{P}_1$.

The abstract upper join *of* $\mathbb{P}_1$ *w.r.t.* $(\lambda, \mathbb{P}_2)$ *is the set of all existing elementary abstract upper joins of* $(sb_1, \pi_1) \in \mathbb{P}_1$ *with* $(sb_2, \pi_2) \in \mathbb{P}_2$, *where* $sb_1, sb_2 \subseteq body(R)$ *for some* $R \in \mathcal{R}$. *It is denoted by* $Join_{\text{u}}(\mathbb{P}_1, \lambda, \mathbb{P}_2)$.

The abstract lower join *of* $\mathbb{P}_2$ *w.r.t.* $(\lambda, \mathbb{P}_1)$ *is the set of all existing elementary abstract lower joins of* $(sb_1, \pi_1) \in \mathbb{P}_1$ *with* $(sb_2, \pi_2) \in \mathbb{P}_2$, *where* $sb_1, sb_2 \subseteq body(R)$ *for some* $R \in \mathcal{R}$. *It is denoted by* $Join_{\text{l}}(\mathbb{P}_1, \lambda, \mathbb{P}_2)$.

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

Property 12. *If* $\mathbb{P}_1 \rightsquigarrow \lambda.\mathbb{P}_2$ *is a sound creation rule, then so is* $\mathbb{P}_1 \rightsquigarrow \lambda.Join_{\text{l}}(\mathbb{P}_1, \lambda, \mathbb{P}_2)$.

Proof. Let S be a derivation, B_1 be a bag of $(DT(S), P)$ such that $\mathbb{P}_1 \sqsubseteq P(B_1)$. Since $\mathbb{P}_1 \rightsquigarrow \lambda.\mathbb{P}_2$ is sound, there are a derivation S' with patterned derivation tree $(DT(S'), P')$ and a child B_2 of B_1 in S' linked to B_1 by λ such that $\mathbb{P}_2 \sqsubseteq P'(B_2)$. By soundness of join propagation, $Join(P'(B_2), P'(B_1)) \sqsubseteq P'(B_2)$. By monotonicity of the join operation, we obtain that $\mathbb{P}_1 \rightsquigarrow \lambda.Join_{\text{l}}(\mathbb{P}_1, \lambda, \mathbb{P}_2))$ is a sound rule. □

Property 13. *If $\mathbb{P}_1 \rightsquigarrow \lambda.\mathbb{P}_2$ is a sound creation rule, then $\mathbb{P}_1 \rightsquigarrow \mathit{Join}_{\mathrm{u}}(\mathbb{P}_1, \lambda, \mathbb{P}_2)$ is a sound evolution rule.*

Proof. Similar to the proof of Prop 12. □

Property 14. *If $\mathbb{P}_1 \rightsquigarrow \mathbb{P}_2$ and $\mathbb{P}_2 \rightsquigarrow \mathbb{P}_3$ are sound evolution rules, then $\mathbb{P}_1 \rightsquigarrow \mathbb{P}_3$ is also a sound evolution rule.*

Proof. Let S be a derivation, and let B be a bag of $(DT(S), P)$ such that $\mathbb{P}_1 \sqsubseteq P(B)$. Since $\mathbb{P}_1 \rightsquigarrow \mathbb{P}_2$ is sound, there exists a derivation S' extending S such that $\mathbb{P}_2 \sqsubseteq P'(B)$. Since $\mathbb{P}_2 \rightsquigarrow \mathbb{P}_3$ is sound, there exists a derivation S'' extending S' such that $\mathbb{P}_3 \sqsubseteq P''(B)$. Since S'' is also a derivation extending S, it holds that $\mathbb{P}_1 \rightsquigarrow \mathbb{P}_3$ is sound. □

Property 15. *If $\mathbb{P}_1 \rightsquigarrow \mathbb{P}_2$ and $\mathbb{P}_1 \rightsquigarrow \lambda.\mathbb{P}_3$ are sound evolution/creation rules, then $\mathbb{P}_2 \rightsquigarrow \lambda.\mathbb{P}_3$ is a sound creation rule.*

Proof. The property holds by monotonicity of the join operation, and by the condition that $\mathbb{P}_1 \rightsquigarrow \mathbb{P}_2$ being sound implies $\mathbb{P}_1 \sqsubseteq \mathbb{P}_2$. □

Property 16. *If $\mathbb{P}_1 \rightsquigarrow \lambda.\mathbb{P}_2$ and $\mathbb{P}_2 \rightsquigarrow \mathbb{P}_3$ are sound creation/evolution rules, then $\mathbb{P}_1 \rightsquigarrow \lambda.\mathbb{P}_3$ is a sound creation rule.*

Proof. Let S be a derivation, and let B_1 be a bag of $(DT(S), P)$ such that $\mathbb{P}_1 \sqsubseteq P(B_1)$. Since $\mathbb{P}_1 \rightsquigarrow \lambda.\mathbb{P}_2$ is sound, there are a derivation S' with patterned derivation tree $(DT(S'), P')$ and a child B_2 of B_1 in S' that is linked to B_1 by λ such that $\mathbb{P}_2 \sqsubseteq P'(B_2)$. Since $\mathbb{P}_2 \rightsquigarrow \mathbb{P}_3$ is sound, there exists a derivation S'' extending S' such that $\mathbb{P}_3 \sqsubseteq P''(B_2)$. S'' extends S as well, and thus $\mathbb{P}_1 \rightsquigarrow \lambda.\mathbb{P}_3$ is a sound creation rule. □

We call *(abstract) pattern saturation* the already outlined procedure that builds all creation and evolution rules w.r.t. F and $\mathcal{R}$, obtained via an exhaustive application of all deduction rules displayed in Fig. 5. We now prove that this process terminates.

Property 17 (Termination)**.** *For any fact F and any $\mathcal{gbts}$ set of rules $\mathcal{R}$, abstract pattern saturation terminates.*

Proof. There is a finite number of abstract patterns and links between them, 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. □

For technical purposes, we will use the *rank* of an evolution/creation rule.

Definition 32 (Rank)**.** *The* rank *of an evolution or a creation rule is the minimum number of deduction rules (Figure 5) necessary to derive that rule.*

This notion of rank helps us to prove the following technical lemma, that states that the pattern saturation respects some notion of monotonicity: at least as much information can be derived from an abstract pattern $\mathbb{P}_2$ as from an abstract pattern $\mathbb{P}_1$ if $\mathbb{P}_1 \sqsubseteq \mathbb{P}_2$.

Lemma 18. *Let $\mathbb{P}_1$ and $\mathbb{P}_2$ be two abstract patterns such that $\mathbb{P}_1 \sqsubseteq \mathbb{P}_2$. For any rule $\mathbb{P}_1 \rightsquigarrow \mathbb{P}'_1$ (resp. $\mathbb{P}_1 \rightsquigarrow \lambda.\mathbb{P}'_1$) in the pattern saturation, there exists a rule $\mathbb{P}_2 \rightsquigarrow \mathbb{P}'_2$ (resp. $\mathbb{P}_2 \rightsquigarrow \lambda.\mathbb{P}'_2$) in the pattern saturation such that $\mathbb{P}'_1 \sqsubseteq \mathbb{P}'_2$.*

$$\frac{}{\mathbb{P} \rightsquigarrow \pi_{|\sigma(fr(R))}.\mathbb{P}_{\text{init}}(\mathbb{B}(R,\sigma))}\ \text{Prop. 11}$$

$$\frac{\mathbb{P}_1 \rightsquigarrow \lambda.\mathbb{P}_2}{\mathbb{P}_1 \rightsquigarrow \lambda.\mathit{Join}_{\text{l}}(\mathbb{P}_1,\lambda,\mathbb{P}_2)}\text{Prop. 12} \qquad \frac{\mathbb{P}_1 \rightsquigarrow \lambda.\mathbb{P}_2}{\mathbb{P}_1 \rightsquigarrow \mathit{Join}_{\text{u}}(\mathbb{P}_1,\lambda,\mathbb{P}_2)}\text{Prop. 13}$$

$$\frac{\mathbb{P}_1 \rightsquigarrow \mathbb{P}_2 \quad \mathbb{P}_2 \rightsquigarrow \mathbb{P}_3}{\mathbb{P}_1 \rightsquigarrow \mathbb{P}_3}\text{Prop. 14}$$

$$\frac{\mathbb{P}_1 \rightsquigarrow \mathbb{P}_2 \quad \mathbb{P}_1 \rightsquigarrow \lambda.\mathbb{P}_3}{\mathbb{P}_2 \rightsquigarrow \lambda.\mathbb{P}_3}\text{Prop. 15} \qquad \frac{\mathbb{P}_1 \rightsquigarrow \lambda.\mathbb{P}_2 \quad \mathbb{P}_2 \rightsquigarrow \mathbb{P}_3}{\mathbb{P}_1 \rightsquigarrow \lambda.\mathbb{P}_3}\text{Prop. 16}$$

Figure 5: Deduction calculus for the pattern saturation rules. For the first deduction rule, $R \in \mathcal{R} \cup \{\rightarrow F\}$, π is a homomorphism from $\mathit{fr}(R)$ to $\mathit{terms}(\mathit{support}(\mathbb{P}))$ such that $(\mathit{body}(R),\pi) \in \mathbb{P}$, and σ is the fusion of $\mathit{fr}(R)$ induced by π.

Proof. We prove the result by induction on the rank of $\mathbb{P}_1 \rightsquigarrow \mathbb{P}'_1$ (resp. $\mathbb{P}_1 \rightsquigarrow \lambda.\mathbb{P}'_1$). At rank 0, the result is vacuously true.

- Let $\mathbb{P}_1 \rightsquigarrow \mathbb{P}'_1$ be a rule of rank n of the pattern saturation. It has been obtained by applying Property 13 or Property 14 to rules of rank strictly smaller than n. Let us first consider that Property 13 has been applied. Let $\mathbb{P}_1 \rightsquigarrow \lambda.\tilde{\mathbb{P}_1}$ be the rule on which Property 13 has been applied. By induction hypothesis, there exists a rule $\mathbb{P}_2 \rightsquigarrow \lambda.\tilde{\mathbb{P}_2}$ in the pattern saturation such that $\tilde{\mathbb{P}_1} \sqsubseteq \tilde{\mathbb{P}_2}$. By monotonicity of the join operation, it holds that $\mathbb{P}'_1 = \mathit{Join}_{\text{u}}(\mathbb{P}_1,\lambda,\tilde{\mathbb{P}_1}) \sqsubseteq \mathit{Join}_{\text{u}}(\mathbb{P}_2,\lambda,\tilde{\mathbb{P}_2}) = \mathbb{P}'_2$. Thus $\mathbb{P}_2 \rightsquigarrow \mathbb{P}'_2$ is in the pattern saturation and $\mathbb{P}'_1 \sqsubseteq \mathbb{P}'_2$. Let us now consider that Property 14 has been used to create $\mathbb{P}_1 \rightsquigarrow \mathbb{P}'_1$. Then, there are two rules of rank strictly smaller than n, namely $\mathbb{P}_1 \rightsquigarrow \mathbb{P}''_1$ and $\mathbb{P}''_1 \rightsquigarrow \mathbb{P}'_1$. By induction hypothesis, there is a rule $\mathbb{P}_2 \rightsquigarrow \mathbb{P}''_2$ in the pattern saturation such that $\mathbb{P}''_1 \sqsubseteq \mathbb{P}''_2$. We can once again apply the induction hypothesis, and we conclude that there exists a rule $\mathbb{P}''_2 \rightsquigarrow \mathbb{P}'_2$ in the pattern saturation, where $\mathbb{P}'_1 \sqsubseteq \mathbb{P}'_2$. By applying Property 14, we conclude that $\mathbb{P}_2 \rightsquigarrow \mathbb{P}'_2$ is in the pattern saturation.
- Let $\gamma_e : P_1 \rightsquigarrow \lambda.\mathbb{P}'_1$ be a rule of rank n of the pattern saturation. It may have been created by application of Properties 11, 12, 15 or 16.
  - If γ_e has been created by Property 11, then the rule $\mathbb{P}_2 \rightsquigarrow \lambda.\mathbb{P}'_1$ can also be created thanks to this property, since $\mathbb{P}_1 \sqsubseteq \mathbb{P}_2$.
  - If γ_e has been created by Property 12, then there is a rule $\mathbb{P}_1 \rightsquigarrow \lambda.\mathbb{P}''_1$ of rank strictly smaller than n in the pattern saturation, which has been used to create γ_e. By induction hypothesis, there is a rule $\mathbb{P}_2 \rightsquigarrow \lambda.\mathbb{P}''_2$. We define $\mathbb{P}'_2 = \mathit{Join}_{\text{l}}(\mathbb{P}_2,\lambda,\mathbb{P}''_2)$. By monotonicity of the join operation, we have that $\mathbb{P}'_1 = \mathit{Join}_{\text{l}}(\mathbb{P}_1,\lambda,\mathbb{P}''_1) \sqsubseteq \mathit{Join}_{\text{l}}(\mathbb{P}_2,\lambda,\mathbb{P}''_2) = \mathbb{P}'_2$. By applying Property 12, we create $\mathbb{P}_2 \rightsquigarrow \lambda.\mathbb{P}'_2$, which shows the claim.
  - If γ_e has been created by Property 15, there are two rules $\mathbb{P}''_1 \rightsquigarrow \mathbb{P}_1$ and $\mathbb{P}''_1 \rightsquigarrow \lambda.\mathbb{P}'_1$ of rank strictly less than n in the pattern saturation. Since $\mathbb{P}''_1 \sqsubseteq \mathbb{P}_1 \sqsubseteq \mathbb{P}_2$, we

can directly apply the induction hypothesis and state the existence of $\mathbb{P}_2 \rightsquigarrow \lambda.\mathbb{P}'_2$ with $\mathbb{P}'_1 \sqsubseteq \mathbb{P}'_2$.

– If γ_e has been created by Property 16, there exists two rules $\mathbb{P}_1 \rightsquigarrow \lambda.\mathbb{P}''_1$ and $\mathbb{P}''_1 \rightsquigarrow \mathbb{P}'_1$ of rank strictly less than n in the pattern saturation. By induction hypothesis, there exists a rule $\mathbb{P}_2 \rightsquigarrow \lambda.\mathbb{P}''_2$ in the pattern saturation, with $\mathbb{P}''_1 \sqsubseteq \mathbb{P}''_2$. We can once again apply the induction hypothesis, inferring the existence of $\mathbb{P}''_2 \rightsquigarrow \mathbb{P}'_2$ in the pattern saturation, with $\mathbb{P}'_1 \sqsubseteq \mathbb{P}'_2$. By applying Property 16, we infer the existence of $\mathbb{P}_2 \rightsquigarrow \lambda.\mathbb{P}'_2$, which concludes the proof.

□

In the obtained fixpoint, some rules are redundant. For instance, if there exist two rules $\mathbb{P}_1 \rightsquigarrow \lambda.\mathbb{P}_2$ and $\mathbb{P}_1 \rightsquigarrow \lambda.\mathbb{P}_3$, with $\mathbb{P}_2 \sqsubseteq \mathbb{P}_3$, then the first rule is implied by the second one. This motivate the definition of *most informative rules*.

Definition 33 (Most informative rules)**.** *Let F be a fact and $\mathcal{R}$ be a gbts set of rules. The set of most informative rules associated with F and $\mathcal{R}$, denoted by $\mathcal{I}_{F,\mathcal{R}}$ is the maximal subset of the abstract pattern saturation such that:*

- *a creation rule $\mathbb{P}_1 \rightsquigarrow \lambda.\mathbb{P}_2$ belongs to $\mathcal{I}_{F,\mathcal{R}}$ if there is no rule in the abstract $\mathbb{P}_1 \rightsquigarrow \lambda.\mathbb{P}_2$ with $\mathbb{P}_2 \neq \mathbb{P}_3$ and $\mathbb{P}_2 \sqsubseteq \mathbb{P}_3$;*
- *an evolution rule $\mathbb{P}_1 \rightsquigarrow \mathbb{P}_2$ belongs to $\mathcal{I}_{F,\mathcal{R}}$ if there is no rule $\mathbb{P}_1 \rightsquigarrow \mathbb{P}_2$ in the abstract pattern saturation that satisfies $\mathbb{P}_2 \neq \mathbb{P}_3$ and $\mathbb{P}_2 \sqsubseteq \mathbb{P}_3$.*

Let us notice that we can without ambiguity speak about *the* evolution rule of the most informative rule set having a given abstract pattern as left hand side (when it exists), as there is at most one such rule.

Property 19. *Let F be a fact, $\mathcal{R}$ be a gbts set of rules, and $\mathbb{P}$ be an abstract pattern. $\mathcal{I}_{F,\mathcal{R}}$ contains at most one evolution rule and at most one creation rule having $\mathbb{P}$ as left-hand side.*

Proof. We show that if $\mathbb{P} \rightsquigarrow \lambda.\mathbb{P}_1$ and $\mathbb{P} \rightsquigarrow \lambda.\mathbb{P}_2$ (resp. $\mathbb{P} \rightsquigarrow \mathbb{P}_1$ and $\mathbb{P} \rightsquigarrow \mathbb{P}_2$) belong to the pattern saturation, then there exists $\mathbb{P}_3$ with $\mathbb{P}_1 \sqsubseteq \mathbb{P}_3$ and $\mathbb{P}_2 \sqsubseteq \mathbb{P}_3$ such that $\mathbb{P} \rightsquigarrow \lambda.\mathbb{P}_3$ (resp. $\mathbb{P} \rightsquigarrow \mathbb{P}_3$) belongs to the pattern saturation as well.

We assume without loss of generality that $\mathbb{P} \rightsquigarrow \lambda.\mathbb{P}_1$ (resp. $\mathbb{P} \rightsquigarrow \mathbb{P}_1$) is the rule of smallest rank, and we prove the result by induction on that rank. A rule can be of rank 1 if and only if it has been created thanks to Property 11. Since the only way to create a rule of the shape $\mathbb{P} \rightsquigarrow \lambda.\mathbb{P}_1$ is to use that property and that other deduction rules may only make the patterns grow, it holds that $\mathbb{P}_1 \sqsubseteq \mathbb{P}_2$, and we can take $\mathbb{P}_3 = \mathbb{P}_2$. The results thus holds if the first rule is of rank 1. Let us assume the result to be true for any rule up to rank n, and let us show that it is true as well for any rule of rank $k+1$. We first consider creation rules and we distinguish three cases:

- $\mathbb{P} \rightsquigarrow \lambda.\mathbb{P}_1$ has been created by Property 12. We can apply the induction hypothesis on the premises of that deduction rule, say $\mathbb{P} \rightsquigarrow \lambda.\mathbb{P}'_1$. There exists thus $\mathbb{P}'_3$ such that $\mathbb{P} \rightsquigarrow \lambda.\mathbb{P}'_3$ belongs to the pattern saturation and $P'_1 \sqsubseteq P'_3$ and $P_2 \sqsubseteq P'_3$. By applying then Property 12, and by monotonicity of the join operation, one get $\mathbb{P} \rightsquigarrow \lambda.\mathbb{P}_3$ as desired.

- $\mathbb{P} \rightsquigarrow \lambda.\mathbb{P}_1$ has been created by Property 15. We apply the induction hypothesis on the premise that is a creation rule, which allows us to conclude.
- $\mathbb{P} \rightsquigarrow \lambda.\mathbb{P}_1$ has been created by Property 16. We apply the induction hypothesis on the premise that is a creation rule; Lemma 18 then allows us to conclude.

We now consider evolution rules. We distinguish two cases:

- $\mathbb{P} \rightsquigarrow \mathbb{P}_1$ has been created by Proposition 13. As in the first case of the creation rules, the result follow by induction hypothesis and monotonicity of the join operation.
- $\mathbb{P} \rightsquigarrow \mathbb{P}_1$ has been created by Proposition 14. The result follow by induction hypothesis on the first premise and by Lemma 18.

□

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 16. *The initial pattern $\mathbb{P}_F$ of F_{ex} (Example 7) contains the following elements:*$(q_1(x_1, y_1, z_1), \{x_1 \mapsto a, y_1 \mapsto b, z_1 \mapsto c\})$, $(q_1(x_1, y_1, z_1), \{x_1 \mapsto d, y_1 \mapsto c, z_1 \mapsto e\})$ *and* $(q_1(x_1, y_1, z_1), \{x_1 \mapsto f, y_1 \mapsto g, z_1 \mapsto g\}$. *By application of Property 11, three novel rules are created: $\mathbb{P}_0 \rightsquigarrow \emptyset.\mathbb{P}_1^{b,c}$, $\mathbb{P}_0 \rightsquigarrow \emptyset.\mathbb{P}_1^{c,e}$ and $\mathbb{P}_0 \rightsquigarrow \emptyset.P_1^{g,g}$, where $\mathbb{P}_1^{b,c}, P_1^{c,e}$, and $\mathbb{P}_1^{g,g}$ are described below.*

The atoms of the abstract bag associated with $\mathbb{P}_1^{b,c}$ are $\{s(b, t_1), r(c, t_1), q_2(t_1, u_1, v_1)\}$, and its link is empty (since the whole frontier of R_1^{ex} is mapped to constants). The atoms of the abstract bag associated with $\mathbb{P}_1^{c,e}$ are $\{s(c, t_1), r(e, t_1), q_2(t_1, u_1, v_1)\}$, and those of the abstract bag associated with $\mathbb{P}_1^{g,g}$ are $\{s(g, t_1), r(g, t_1), q_2(t_1, u_1, v_1)\}$.

$\mathbb{P}_1^{b,c}$ contains the following pairs:

- $(\{q_2(x_2, y_2, z_2)\}, \{x_2 \mapsto t_1, y_2 \mapsto u_1, z_2 \mapsto v_1\})$;
- $(\{q_2(x_4, y_4, z_4)\}, \{x_4 \mapsto t_1, y_4 \mapsto u_1, z_4 \mapsto v_1\})$;
- $(\{s(y_4, t_4)\}, \{y_4 \mapsto b, t_4 \mapsto t_1\})$;
- $(\{r(z_4, t_4)\}, \{z_4 \mapsto c, t_4 \mapsto t_1\})$;
- $(\{s(y_4, t_4), r(z_4, t_4)\}, \{y_4 \mapsto b, z_4 \mapsto c, t_4 \mapsto t_1\})$;
- $(\{s(y_5, t_5)\}, \{y_5 \mapsto b, t_5 \mapsto t_1\})$;
- $(\{r(z_5, t_5)\}, \{z_5 \mapsto c, t_5 \mapsto t_1\})$;
- $(\{s(y_5, t_5), r(z_5, t_5)\}, \{y_5 \mapsto b, z_5 \mapsto c, t_5 \mapsto t_1\})$.

$\mathbb{P}_1^{c,e}$ contains the same pairs, except that every occurrence of b is replaced by c and every occurrence of c is replaced by e, whereas $\mathbb{P}_1^{g,g}$ contains the same pairs, except that every occurrence of b is replaced by g and every occurrence of c is replaced by g.

$\mathbb{P}_1^{b,c}$ is graphically represented in Figure 6.

These three patterns contain $(\{q_2(x_2, y_2, z_2)\}, \{x_2 \mapsto t_1, y_2 \mapsto u_1, z_2 \mapsto v_1\})$, and we thus create the three following rules:

Atoms of the abstract bag:
$s(b, t_1), r(c, t_1), q_2(t_1, u_1, v_1)$

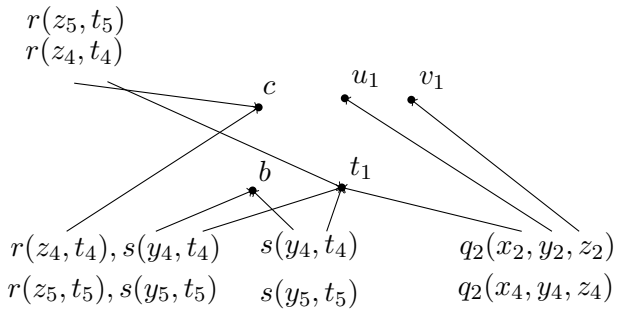

Figure 6: A graphical representation of $\mathbb{P}_1^{b,c}$

- $\mathbb{P}_1^{b,c} \rightsquigarrow \lambda'.\mathbb{P}_2$,
- $\mathbb{P}_1^{c,e} \rightsquigarrow \lambda'.\mathbb{P}_2$,
- $\mathbb{P}_1^{g,g} \rightsquigarrow \lambda'.\mathbb{P}_2$,

where $\lambda' = \{y_2 \mapsto u_1, z_2 \mapsto v_1\}$ *and* $\mathbb{P}_2$ *is defined below.*
The atoms of $\mathbb{P}_2$ *are* $\{s(y_2, t_2), r(z_2, t_2), q_3(t_2, u_2, v_2)\}$*. It contains the following elements:*

- $(\{q_3(t_3, u_3, v_3)\}, \{t_3 \mapsto t_2, u_3 \mapsto u_2, v_3 \mapsto v_2\})$,
- $(\{s(y_4, t_4)\}, \{y_4 \mapsto y_2, t_4 \mapsto t_2\})$,
- $(\{r(z_4, t_4)\}, \{z_4 \mapsto z_2, t_4 \mapsto t_2\})$,
- $(\{s(y_4, t_4), r(z_4, t_4)\}, z_4 \mapsto z_2, y_4 \mapsto y_2, t_4 \mapsto t_2\})$,
- $(\{s(y_5, t_5)\}, \{y_5 \mapsto y_2, t_5 \mapsto t_2\})$,
- $(\{r(z_5, t_5)\}, \{z_5 \mapsto z_2, t_5 \mapsto t_2\})$,
- $(\{s(y_5, t_5), r(z_5, t_5)\}, \{y_5 \mapsto y_2, z_5 \mapsto z_2, t_5 \mapsto t_2\})$.

The element $(\{q_3(t_3, u_3, v_3)\}, \{t_3 \mapsto t_2, u_3 \mapsto u_2, v_3 \mapsto v_2\})$ *belongs to* $\mathbb{P}_2$*, and thus, we create a rule* $\mathbb{P}_2 \rightsquigarrow \lambda''.\mathbb{P}_3$*), where* $\lambda'' = \{t_3 \mapsto t_2\}$ *and* $\mathbb{P}_3$ *contains the following elements:*

- $(\{h(t_4)\}, \{t_4 \mapsto t_2\})$,
- $(\{h(t_5)\}, \{t_5 \mapsto t_2\})$.

At this point, we cannot create any new rule by Property 11. However, Property 13 may be used to derive an evolution of $\mathbb{P}_2$*. Indeed, since* $\mathbb{P}_2 \rightsquigarrow \lambda''.\mathbb{P}_3$ *has been derived, we can derive* $\mathbb{P}_2 \rightsquigarrow \mathbb{P}_2'$ *with* $\mathbb{P}_2' = \mathit{Join}_{\mathrm{u}}(\mathbb{P}_2, \lambda'', \mathbb{P}_3)$*. Note that* $\mathbb{P}_2'$ *is a superset of* $\mathbb{P}_2$ *that additionally contains the following elements:*

- $(\{s(y_4, t_4), h(t_4)\}, \{y_4 \mapsto y_2, t_4 \mapsto t_2\})$,
- $(\{r(z_4, t_4), h(t_4)\}, \{z_4 \mapsto z_2, t_4 \mapsto t_2\})$,
- $(\{s(y_4, t_4), r(z_4, t_4), h(t_4)\}, \{z_4 \mapsto z_2, y_4 \mapsto y_2, t_4 \mapsto t_2\})$,
- $(\{s(y_5, t_5), h(t_5)\}, \{y_5 \mapsto y_2, t_5 \mapsto t_2\})$,
- $(\{r(z_5, t_5), h(t_5)\}, \{z_5 \mapsto z_2, t_5 \mapsto t_2\})$,
- $(\{s(y_5, t_5), r(z_5, t_5), h(t_5)\}, \{y_5 \mapsto y_2, z_5 \mapsto z_2, t_5 \mapsto t_2\})$,
- $(\{h(t_4)\}, \{t_4 \mapsto t_2\})$,
- $(\{h(t_5)\}, \{t_5 \mapsto t_2\})$.

By Property 16, the following sound rules can then be obtained:

- $\mathbb{P}_1^{b,c} \rightsquigarrow \lambda'.\mathbb{P}'_2$,
- $\mathbb{P}_1^{c,e} \rightsquigarrow \lambda'.\mathbb{P}'_2$,
- $\mathbb{P}_1^{g,g} \rightsquigarrow \lambda'.\mathbb{P}'_2$.

Applying once more Property 13 yields new sound rules such as:

$$\mathbb{P}_1^{b,c} \rightsquigarrow \mathbb{P}_1^{b,c'},$$

where $\mathbb{P}_1^{b,c'}$ is a superset of $\mathbb{P}_1^{b,c}$ that additionally contains, among others, the following element:

$$(\{q_2(x_4, y_4, z_4), r(z_4, t_4), s(y_4, t_4), h(t_4)\}, \{x_4 \mapsto t_1, y_4 \mapsto u_1, z_4 \mapsto v_1\}).$$

Please note that in this case, $\pi = \{x_4 \mapsto t_1, y_4 \mapsto u_1, z_4 \mapsto v_1\}$ 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 π does not belong to the terms relevant to the supporting bag.

We skip a part of the further development of this example. It can be checked that at some point, a rule $\mathbb{P}_F \rightsquigarrow \mathbb{P}'_F$ is created, where $\mathbb{P}'_F$ contains the following elements:

$$(\{p_1(x_p), i(x_p)\}, \{x_p \mapsto g\}) \qquad (\{p_2(x_q), i(x_q)\}, \{x_q \mapsto g\})$$

The following two creation rules are thus sound and relevant:

$$\mathbb{P}'_0 \rightsquigarrow \emptyset.\mathbb{P}^i_p \qquad \mathbb{P}'_0 \rightsquigarrow \emptyset.\mathbb{P}^i_q$$

where $\mathbb{P}^i_p$ contains in particular $(\{p_2(x_q), i(x_q)\}, \{x_q \mapsto y_p\})$ and $\mathbb{P}^i_q$ contains $(\{p_1(x_p), i(x_p)\}, \{x_p \mapsto y_q\})$. Since the body of R_6^{ex} belongs to $\mathbb{P}^i_p$, a new creation rule is added: $\mathbb{P}^i_p \rightsquigarrow \{x_p \mapsto y_q\}.\mathbb{P}_q$.

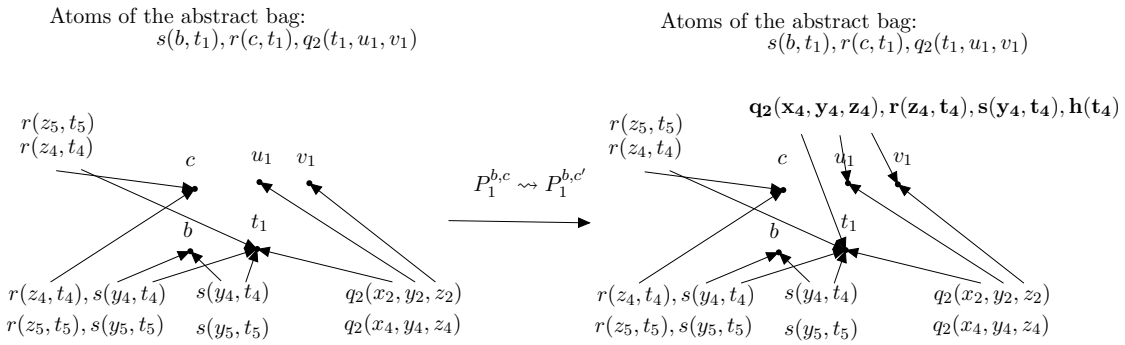

Figure 7: Graphical representation of the rule $\mathbb{P}_1^{b,c} \rightsquigarrow \mathbb{P}_1^{b,c'}$. The new element of $\mathbb{P}_1^{b,c'}$ is in bold.

Likewise, since the body of R_7^{ex} *belongs to* $\mathbb{P}_q^i$*, a new creation rule is added:* $\mathbb{P}_q^i \rightsquigarrow \{x_q \mapsto y_p\}.\mathbb{P}_p$.

Last, two recursive rules are added:

$$\mathbb{P}_p \rightsquigarrow \{x_p \mapsto y_q\}.\mathbb{P}_q, \qquad \mathbb{P}_q \rightsquigarrow \{x_q \mapsto y_p\}.\mathbb{P}_p.$$

4.4 Computation of the Full Blocked Tree

Given a fact, a set of *gbts* rules and their associated set of most informative rules, Algorithm 1 outputs a full blocked tree for F and $\mathcal{R}$. We start by creating a bag with set of terms T_0. This bag is the root of the full blocked tree. We maintain a list of blocked patterns: any bag that is of that pattern and that is not labeled as non-blocked is thus blocked. We then consider the most informative evolution rule having $\mathbb{P}_F = \mathbb{P}_{\text{init}}(\mathbb{B}(\rightarrow F, \emptyset))$ (i.e., the initial abstract pattern of F) as left-hand side, say $\mathbb{P}_F \rightarrow \mathbb{P}_F^*$. We label the root of the full blocked tree with the pattern $\mathbb{P}_F^*$.[8] We mark this newly created root as being non-blocked. Then, as long as there exist a non-blocked bag B_1 and a most informative creation rule $\mathbb{P}_1 \rightsquigarrow \lambda.\mathbb{P}_2$ with $\mathbb{P}_1 \sim P(B_1)$ that has not been applied on B_1, we apply that rule. To apply it, we add a child B_2 to B_1 such that $P(B_2) \sim \mathbb{P}_2$ and the induced link from B_2 to B_1 is λ. B_2 is considered blocked (*i.e.*, is not marked non-blocked) if there is already a bag B_3 with $P(B_2) \sim P(B_3)$ in the built structure, and non-blocked otherwise. This procedure halts, since there is a finite number of non-equivalent patterns, and the maximal degree of the built tree is also bounded. It creates a *sound* blocked tree, since all creation and evolution

8. Note that, technically, we abuse a abstract pattern as a non-abstract pattern here, but this is not a problem since no safe renaming is necessary for the (pattern of) bag F.

Algorithm 1: Creation of a full blocked tree

Data: A fact F, a set of *gbts* rules $\mathcal{R}$, the set of most informative rules $\mathcal{I}_{F,\mathcal{R}}$.
Result: $\mathfrak{T}^*_b(F,\mathcal{R})$, a full blocked tree for F and $\mathcal{R}$.
define the root of $\mathfrak{T}^*_b(F,\mathcal{R})$ to be B_F;
assign to B_F a pattern $P_F \sim \mathbb{P}^*_F$, such that $\mathbb{P}_{\text{init}}(\mathbb{B}(\rightarrow F, \emptyset)) \rightsquigarrow \mathbb{P}^*_F \in \mathcal{I}_{F,\mathcal{R}}$;
blocked-patterns := P^*_F;
non-blocked-bags := B_F;
next-non-blocked := $\emptyset$;
while *non-blocked-bags* $\neq \emptyset$ **do**
  next-non-blocked := $\emptyset$;
  for $B_1 \in$ *non-blocked-bags* **do**
    $\mathbb{P}_1$:= abstract pattern of B_1;
    for *all creation rule* $\mathbb{P}_1 \rightsquigarrow \lambda.\mathbb{P}_2 \in \mathcal{I}_{F,\mathcal{R}}$ **do**
      Add in $\mathfrak{T}^*_b(F,\mathcal{R})$ a child B_2 to B_1, with induced link λ;
      Define the pattern of B_2 to be $P_2 \sim \mathbb{P}_2$;
      if $\mathbb{P}_2 \notin$ *blocked-patterns* **then**
        next-non-blocked-bags := next-non-blocked-bags $\cup\{B_2\}$;
        blocked-patterns := blocked-patterns $\cup\{\mathbb{P}_2\}$;
  non-blocked-bags := next-non-blocked;
return $\mathfrak{T}^*_b(F,\mathcal{R})$;

rules are sound. It also creates a *complete* blocked tree, and thus a *full blocked tree*, as will be proven below.

Intuitively, the following property states that for any derivation tree associated with an $\mathcal{R}$-derivation of F, there exists an isomorphic tree generated by $\mathfrak{T}^*_b(F,\mathcal{R})$.

Property 20. *Let F be a fact, $\mathcal{R}$ be gbts. Let S be an $\mathcal{R}$-derivation of F and let $(DT(S),P)$ be the according patterned derivation tree with root B^{root}. Let $\mathfrak{T}^*_b(F,\mathcal{R})$ be the corresponding full blocked tree with pattern-assigning function $P_{\mathfrak{T}^*_b}$ and root $B^{\text{root}}_{\mathfrak{T}^*_b}$.*
Then there exists a tree decomposition $\mathfrak{T}$ generated from $\mathfrak{T}^*_b(F,\mathcal{R})$ via a mapping f, such that there exists a bijection g from the bags of $(DT(S),P)$ to the bags of $\mathfrak{T}$ that satisfies the following conditions:

1. *$g(B^{\text{root}}) = B^{\text{root}}_{\mathfrak{T}^*_b}$, i.e., g maps the root of $(DT(S),P)$ to the root of $\mathfrak{T}^*_b(F,\mathcal{R})$;*
2. *$P(B) \sqsubseteq P_{\mathfrak{T}^*_b}(f(g(B)))$ for all bags B of $(DT(S),P)$;*
3. *for all bags B, B' of $(DT(S),P)$ for which B' is a child of B with induced link λ, $g(B')$ is a child of $g(B)$ with induced link λ.*

Proof. We prove the property by induction on the length of S.

- If S is the empty derivation, its derivation tree is restricted to a single bag labeled by F. Such a tree can be generated from $\mathfrak{T}^*_b(F,\mathcal{R})$, and the pattern of $B^{\text{root}}_{\mathfrak{T}^*_b}$ is the root of $\mathfrak{T}$, is by construction greater than the initial pattern of the original fact.

- Let us assume that the property is true for all derivations of length $n \geq 0$, and let us show that it also holds for any derivation of length $n+1$. Let S be a derivation of length $n+1$, and let S_n be its restriction to the n first rule applications. Let B_{n+1} be the bag newly created in $DT(S)$, and B_n its parent. By induction hypothesis, there exist g, $\mathfrak{T}_n$ and f_n fulfilling the conditions from 1 to 3 for S_n. Let us consider $f_n(B_n)$. By condition 2, we know that $P(f_n(g_n(B_n)))$ is greater than $P(B_n)$. By Lemma 18, $f_n(g_n(B_n))$ has a child B^*_{n+1} whose pattern includes that of B_{n+1} and has induced link λ with $f_n(g_n(B_n))$. By definition of a tree generated from a blocked tree, it holds that $\mathfrak{T}_{n+1}$ can be generated from $\mathfrak{T}^*_b(F, \mathcal{R})$ via f_{n+1}, where:

  - $\mathfrak{T}_{n+1}$ is obtained from $\mathfrak{T}_n$ by copying B^*_{n+1} under $g(B_n)$; we additionally define this bag as being $g(B_{n+1})$;
  - f_{n+1} is obtained by extending f with $f_{n+1}(g(B_{n+1})) = B^*_{n+1}$.

  By induction hypothesis, it holds that $g(B^{\text{root}}) = B^{\text{root}}_{\mathfrak{T}}$; Condition 1 is thus fulfilled. By construction of $g(B_{n+1})$, Condition 3 also. It remains to check Condition 2. This is not trivial, since the patterns of a bag in the fact associated with S_n and with S may be non-equivalent (*i.e.*, the pattern may have "grown"). Let us assume that there exists a bag B^* such that $P(B^*) \not\sqsubseteq P_{\mathfrak{T}^*_b}(f(g(B^*)))$. Let us moreover assume that B^* is (one of) the closest such bag to B_{n+1}. Let us first notice that it cannot be B_{n+1}. Indeed, $P(B_{n+1})$ is obtained by performing a join operation between its initial pattern and $P_{S_n}(B_n)$. By induction hypothesis, $P(B_n) \sqsubseteq P_{\mathfrak{T}^*_b}(f(g(B_n)))$. Thus, by Properties 11 and 12, the pattern saturation contains a rule allowing to create a child of $g(B_n)$ whose pattern includes $P_S(B_{n+1})$ and having induced link λ. Thus B^* is not B_{n+1}, and by Property 9, $P(B^*)$ is obtained by performing a join between $P_{S_n}(B^*)$ and $P(B^*_k)$, where B^*_k is the unique bag on the path from B_{n+1} to B^* that is either a child or a parent of B^*. Let us consider the case where B^*_k is a parent of B^* (the other case is similar). By induction hypothesis, $P_{S_n}(B^*) \sqsubseteq P_{\mathfrak{T}^*_b}(f(g(B^*)))$. By choice of B^*, $P(B^*_k) \sqsubseteq P_{\mathfrak{T}^*_b}(f(g(B^*_k)))$. By construction of $\mathfrak{T}^*_b(F, \mathcal{R})$ and of its generated tree, there is a rule $\mathbb{P}_1 \rightsquigarrow \lambda.\mathbb{P}_2$ in the pattern saturation with $\mathbb{P}_1 \sim P_{\mathfrak{T}^*_b}(f(g(B^*_k)))$ and $\mathbb{P}_2 \sim P_{\mathfrak{T}^*_b}(f(g(B^*)))$. Moreover, since this rule is a most informative rule (by construction of $\mathfrak{T}^*_b(F, \mathcal{R})$), $\mathit{Join}_{\mathrm{l}}(\mathbb{P}_1, \lambda, \mathbb{P}_2) \sqsubseteq P(f(g(B^*)))$. However, by monotonicity of the join operation, this would imply that $P(B^*) \sqsubseteq P_{\mathfrak{T}^*_b}(f(g(B^*)))$, hence a contradiction.

□

By preceding observations and Property 20, we are now able to state that Algorithm 1 is correct, as expressed by the next theorem.

Theorem 21. *Algorithm 1 outputs a full blocked tree.*

Before turning to the more involved querying operation, let us stress 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 - whose 2ExpTime combined complexity was already shown (Calì et al., 2008), are a particular case of *gbts* rules.

Theorem 22. BCQ-Entailment *for* $\mathcal{gbts}$ *is in* 2ExpTime *for combined complexity and in* ExpTime *for data complexity.*

Proof. Let us recall that $F, \mathcal{R} \models Q$ holds exactly if $F, \mathcal{R} \cup \{Q \to match\} \models match$, where $match$ is a fresh, nullary predicate. Note that $\mathcal{R} \cup \{Q \to match\}$ is still $\mathcal{gbts}$ since $Q \to match$ is $\mathcal{fg}$. $F, \mathcal{R} \cup \{Q \to match\} \models match$ can be easily checked given $\mathfrak{T}_b^*(F, \mathcal{R} \cup \{Q \to match\})$ by checking if any of the abstract patterns associated to any of the bags contain some pattern (Q, π) for some π. 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 degree of a derivation tree. The number of patterns is doubly exponential in the data in the worst-case, while the degree is at most exponential. When the rule set (including the query) is fixed, the data complexity falls to ExpTime. Lower bounds for data complexity come from already known complexity results of weakly-guarded rules (Calì et al., 2008), for instance. □

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

4.5 Querying the Full Blocked Tree

We considered in previous sections the query to be implemented via a rule. This trick allowed to have a conceptually easy querying operation, because it 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 expressed via a rule. The second drawback of taking the query into account while building the full blocked tree is that it may prevent us from adapting this construction when assumptions are made on the set of rules: can we devise a better algorithm if we have additional knowledge concerning the rule set, for instance, if we know that it is guarded, and not only $\mathcal{gbts}$?

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 arbitrarily deep patterned derivation trees for F and $\mathcal{R}$, but we cannot just check if a bag is labeled by the query – since the query does not necessarily appear in the considered patterns anymore. A simple homomorphism check is not sufficient either, as can be seen in Example 17 below. To overcome this problem, we introduce a structure called *atom-term partition tree* (APT). Such a structure is meant to encode a decomposition 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 4.5.1. 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

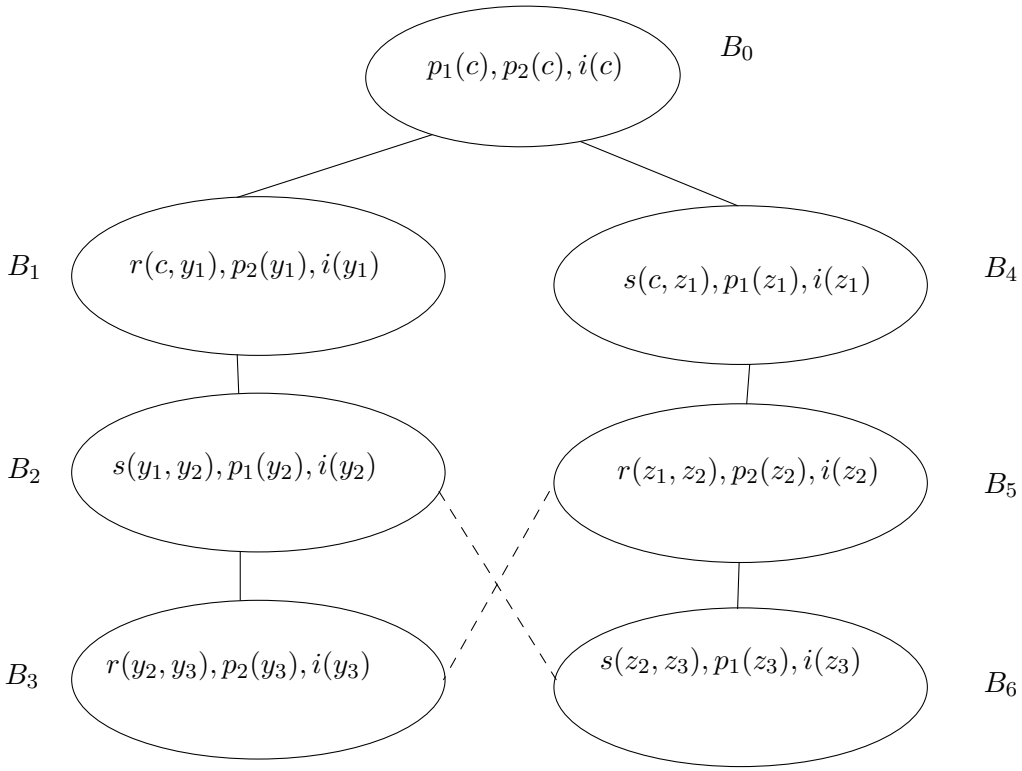

Figure 8: The full blocked tree associated with $F^{ex'}$ and $\mathcal{R}^{ex'}$. B_2 and B_6 are equivalent, as well as B_3 and B_5.

trees, but in the *finite* full blocked tree. In that case, APTs will still be used, but we will have to adjust their validation process (Section 4.5.2).

Let us first stress why the usual homomorphism check is not a suitable operation for querying a full blocked tree. To simplify the presentation, we will restrict the running example in the following way: we only consider rules $R_1^{ex'} = p_1(x_p) \wedge i(x_p) \rightarrow r(x_p, y_p) \wedge p_2(y_p) \wedge i(y_p)$ and $R_2^{ex'} = p_2(x_q) \wedge i(x_q) \rightarrow s(x_q, y_q) \wedge p_1(y_q) \wedge i(y_q)$ (this set will be denoted by $\mathcal{R}^{ex'}$), and the initial fact is restricted to $i(c) \wedge p_1(c) \wedge p_2(c)$ (denoted by $F^{ex'}$).

Example 17. *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 $\mathcal{R}^{ex'}$ and $F^{ex'}$ and displayed in Figure 8, we do not find any answer to this query. However, B_2 is equivalent to B_6, and by considering a derivation tree where B_3 would have a corresponding bag below B_6 (as B_7 in Figure 9), one would find a (correct) mapping of Q_i.

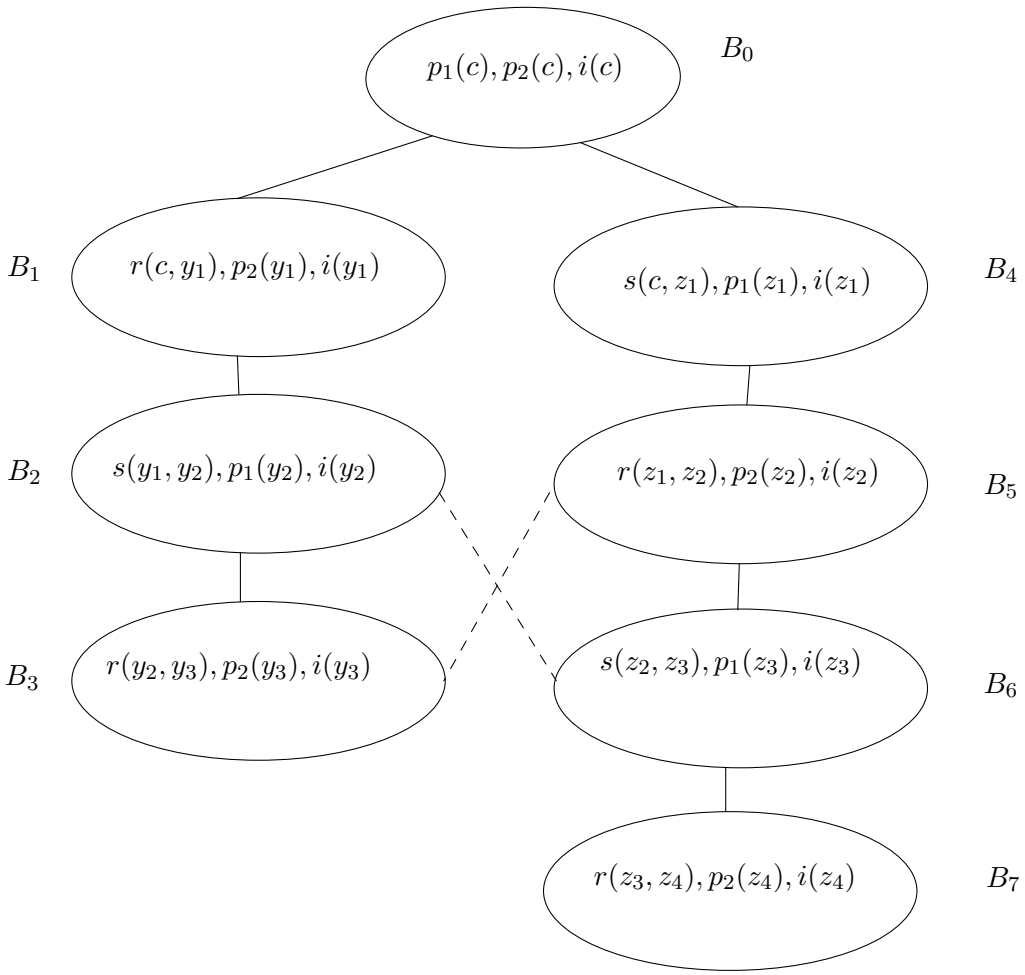

Figure 9: A tree generated by the full blocked tree of Figure 8. B_7 is a copy of B_3 under B_6.

4.5.1 Validation of an APT in a Derivation Tree

Let π be a homomorphism from Q to the atoms of some derivation tree $\mathcal{T} = DT(S)$. From π, let us build an arbitrary mapping $\pi^{\mathrm{a}}_{\mathcal{T}}$ (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 $\mathcal{T}$ with $\pi(a) = p(\pi(t_1), \ldots, \pi(t_k)) \in B$, and define $\pi^{\mathrm{a}}_{\mathcal{T}}(a) = (B, \pi(a))$. Then $\pi^{\mathrm{a}}_{\mathcal{T}}$ gives rise to a partitioning of the atoms of Q into *atom bags* $Bags^{\mathrm{a}}(Q) = \{Q^{\mathrm{a}}_1, \ldots, Q^{\mathrm{a}}_n\}$, where two atoms a and b of Q are in the same atom bag Q^{a}_i if and only if there exists a bag B of T with $\pi^{\mathrm{a}}_{\mathcal{T}}(a) = (B, \pi(a))$ and $\pi^{\mathrm{a}}_{\mathcal{T}}(b) = (B, \pi(b))$.

On another note, it will turn out to be important, given a term t of Q, to keep track of the bag of T in which the term $\pi(t)$ appeared first. We note $\pi^{\mathrm{t}}_{\mathcal{T}}(t) = (B, \pi(t))$ when the term $\pi(t)$ appears first in the bag B of T. Similar to above, $\pi^{\mathrm{t}}_{\mathcal{T}}$ gives rise to a partitioning of the terms of Q into *term bags* $Bags^{\mathrm{t}}(Q) = \{Q^{\mathrm{t}}_1, \ldots, Q^{\mathrm{t}}_m\}$, where two terms u and v of Q are in the same term bag Q^{t}_i if and only if there exists a bag B with $\pi^{\mathrm{t}}_{\mathcal{T}}(u) = (B, \pi(u))$ and $\pi^{\mathrm{t}}_{\mathcal{T}}(v) = (B, \pi(v))$.

From $\pi^{\mathrm{a}}_{\mathcal{T}}$ and $\pi^{\mathrm{t}}_{\mathcal{T}}$, we then obtain the function $\pi_{\mathcal{T}}$ mapping elements of $Bags^{\mathrm{a}}(Q) \cup Bags^{\mathrm{t}}(Q)$ to bags of $\mathcal{T}$ such that

$$\pi_{\mathcal{T}} = \begin{cases} Q^{\mathrm{a}} \mapsto B & \text{where } \pi^{\mathrm{a}}_{\mathcal{T}}(a) = (B, \pi(a)) \text{ for any } a \in Q^{\mathrm{a}} \\ Q^{\mathrm{t}} \mapsto B' & \text{where } \pi^{\mathrm{t}}_{\mathcal{T}}(a) = (B', \pi(t)) \text{ for any } t \in Q^{\mathrm{t}} \end{cases}$$

We can now define the *atom-term* bags of Q (induced by $\pi^{\mathrm{a}}_{\mathcal{T}}$) denoted by $Bags^{\mathrm{at}}(Q)$. If an atom bag Q^{a} and a term bag Q^{t} have the same image under $\pi_{\mathcal{T}}$, we obtain an atom-term bag $Q^{\mathrm{at}} = Q^{\mathrm{a}} \cup Q^{\mathrm{t}}$. If an atom bag Q^{a} (or a term bag Q^{t}) has an image different from the image of any other term bag (or atom bag, respectively) of Q, then it is an atom-term bag by itself.

Finally, we provide these atom-term bags with a tree structure induced by the tree structure of $\mathcal{T}$. Let Q^{at}_1 and Q^{at}_2 be two atom-term bags of Q. Then Q^{at}_2 is a child of Q^{at}_1 iff *(i)* $\pi_{\mathcal{T}}(Q^{\mathrm{at}}_2)$ is a descendant of of $\pi_{\mathcal{T}}(Q^{\mathrm{at}}_1)$ and *(ii)* there is no atom-term bag Q^{at}_3 of Q such that $\pi_{\mathcal{T}}(Q^{\mathrm{at}}_3)$ is a descendant of of $\pi_{\mathcal{T}}(Q^{\mathrm{at}}_1)$ and $\pi_{\mathcal{T}}(Q^{\mathrm{at}}_2)$ is a descendant of of $\pi_{\mathcal{T}}(Q^{\mathrm{at}}_3)$. 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 $\mathcal{T}$ and π.

Definition 34 (APT of a Query)**.** *Let Q be a query. An atom-term partition of Q is a partition of $atoms(Q) \cup terms(Q)$ (these sets being 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 10 represents an APT of the example query Q_i.

Definition 35 ((Valid) APT-Mapping)**.** *Let $\mathcal{Q}$ be an APT of Q. Let $\mathcal{T}$ be a derivation tree. An* APT-mapping *of $\mathcal{Q}$ to $\mathcal{T}$ is a tuple $\Gamma = (\Pi, \pi_1, \ldots, \pi_k)$ where Π is an injective mapping from the atom-term bags of $\mathcal{Q}$ to the bags of $\mathcal{T}$ and, for each atom-term bag Q^{at}_i of $\mathcal{Q}$, π_i is a substitution from the terms of Q^{at}_i (by this, we mean the terms of Q^{at}_i, not the terms appearing in the atoms of Q^{at}_i) to the terms that were created in the atoms of $\Pi(Q^{\mathrm{at}}_i)$.*

Remark that if u is a term of Q, then u appears in only one atom-term bag Q^{at}_i of $\mathcal{Q}$. We can thus define $\pi_\Gamma = \bigcup_{1 \leq i \leq k} \pi_i$.

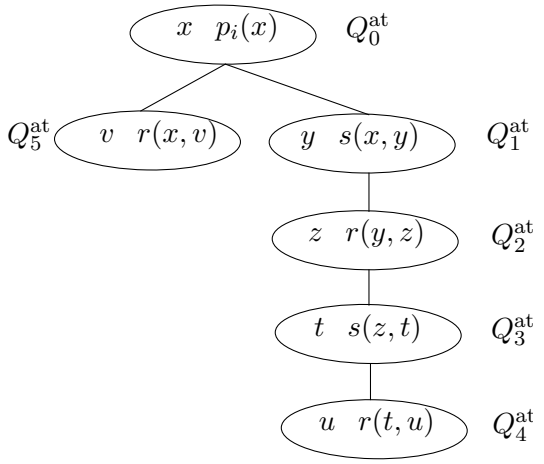

Figure 10: An atom-term partition of $Q_i = p_i(x) \wedge s(x,y) \wedge r(y,z) \wedge s(z,t) \wedge r(t,u) \wedge r(x,v)$

Finally, we say that $(\Pi, \pi_1, \ldots, \pi_k)$ *is* valid *when* π_Γ *is a homomorphism from* Q *to the atoms of* $\mathcal{T}$*.*

Example 18 (APT-Mapping)**.** *We now present a valid APT-Mapping of the APT pictured Figure 10 to the derivation tree represented in Figure 9. We let* $\Pi = \{Q_0^{\text{at}} \mapsto B_0, Q_1^{\text{at}} \mapsto B_4, Q_2^{\text{at}} \mapsto B_5, Q_3^{\text{at}} \mapsto B_6, Q_4^{\text{at}} \mapsto B_7, Q_5^{\text{at}} \mapsto B_1\}$*. The corresponding mappings are:* $\pi_0 = \{x \mapsto c\}$*,* $\pi_1 = \{y \mapsto z_1\}$*,* $\pi_2 = \{z \mapsto z_2\}$*,* $\pi_3 = \{t \mapsto z_3\}$*,* $\pi_4 = \{u \mapsto z_4\}$*, and* $\pi_5 = \{v \mapsto y_1\}$*. Then,* $(\Pi, \pi_1, \pi_2, \pi_3, \pi_4, \pi_5)$ *is a valid APT-mapping of the APT from Figure 10 to the derivation tree from Figure 9.*

Property 23 (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* $\mathcal{Q}$ *of* Q*, and a valid APT-mapping from* $\mathcal{Q}$ *to* $DT(S)$*.*

Proof. We successively prove both directions of the equivalence.

($\Leftarrow$) Let us suppose that there exists a valid APT-mapping from $\mathcal{Q}$ to $DT(S)$. From Definition 35, 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 34, we can choose some mapping $\pi^{\text{a}}_{\mathcal{T}}$ of the atoms of Q, and build from this mapping an APT $\mathcal{Q}$ of Q. We can then build an APT-mapping $(\Pi, \pi_1, \ldots, \pi_k)$ as follows: $\Pi = \pi_{\mathcal{T}}$, and for each Q_i^{at} of $\mathcal{Q}$, π_i is the restriction of π to the terms of Q_i^{at}. 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.

4.5.2 Validation of an APT in a blocked tree

Hence, let us now consider a blocked tree $\mathfrak{T}_b$ and some tree $(\mathfrak{T}, f) \in G(\mathfrak{T}_b)$ generated by it. Let us assume that we have an APT $\mathcal{Q}$ of a query Q that corresponds to a mapping to $(\mathfrak{T}, f)$. Thus, each bag Q_i^{at} of the APT is mapped to a bag B of $\mathfrak{T}$. Intuitively, we represent this mapping on the full blocked tree by mapping Q_i^{at} to B_{rep} such that $B_{\text{rep}} = f(B)$ (*i.e.*, B has been generated by copying B_{rep}). 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 36 (Valid APT Mapping to a Blocked Tree)**.** *Let $\mathcal{Q}$ be an APT of a query Q and $\Gamma = (\Pi, \pi_1, \ldots, \pi_k)$ be an APT-mapping from $\mathcal{Q}$ to a blocked tree $\mathfrak{T}_b$ (where Π maps atom-term bags of $\mathcal{Q}$ to bags of the blocked tree). Then Γ is said to be* valid *if there exists a tree $(\mathfrak{T}, f) \in G(\mathfrak{T}_b)$ generated from $\mathfrak{T}_b$ and a mapping Ξ from the atom-term bags of $\mathcal{Q}$ to the bags of $(\mathfrak{T}, f)$ (we then call $((\mathfrak{T}, f), \Xi)$ a* proof of Γ*) such that:*

- *if Q^{at} is the root of $\mathcal{Q}$, then $\Xi(Q^{\text{at}}) = \Pi(Q^{\text{at}})$;*
- *if $Q^{\text{at}\prime}$ is a child of Q^{at} in $\mathcal{Q}$, then $f(\Xi(Q^{\text{at}\prime})) = \Pi(Q^{\text{at}\prime})$ and $\Xi(Q^{\text{at}\prime})$ is a descendant of $\Xi(Q^{\text{at}})$;*
- *The ADT mapping $(\Xi, \pi'_1, \ldots, \pi'_k)$ is valid in $\mathfrak{T}$, where for every atom-term bag Q_j^{at} in $\mathcal{Q}$ with $\Xi(Q_j^{\text{at}}) = B_i$, we define $\pi'_j = \psi_{f(B_i) \to B_i} \circ \pi_j$.*

Example 19 (APT-Mapping to a Blocked Tree)**.** *We now present a valid APT-Mapping of the APT represented Figure 10 to the derivation tree represented in Figure 9. We define $\Pi = \{Q_0^{\text{at}} \mapsto B_0, Q_1^{\text{at}} \mapsto B_4, Q_2^{\text{at}} \mapsto B_5, Q_3^{\text{at}} \mapsto B_6, Q_4^{\text{at}} \mapsto B_3, Q_5^{\text{at}} \mapsto B_1\}$. Here, the only difference with the previous APT-mapping is the image of Q_4^{at}, which is not B_7 (which does not exist in the blocked tree), but B_3. This is reflected in the definition of the π_i: $\pi_0 = \{x \mapsto c\}$, $\pi_1 = \{y \mapsto z_1\}$, $\pi_2 = \{z \mapsto z_2\}$, $\pi_3 = \{t \mapsto z_3\}$, $\pi_4 = \{u \mapsto y_2\}$, and $\pi_5 = \{v \mapsto y_1\}$. Then, $(\Pi, \pi_1, \pi_2, \pi_3, \pi_4, \pi_5)$ is a valid APT-mapping of the APT from Figure 10 to the blocked tree from Figure 8, as witnessed by the derivation tree of Figure 9, where the bag B_7 has been generated by B_3.*

Property 24 (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 $\mathcal{Q}$ of Q, and a valid APT-mapping from $\mathcal{Q}$ 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 $\mathcal{Q}$ to the full blocked tree $\mathfrak{T}_b$ of F and $\mathcal{R}$. From Definition 36, there exists a valid APT mapping from $\mathcal{Q}$ to a $(\mathfrak{T}, f)$ generated from $\mathfrak{T}_b$, *i.e.*, by Definition 22 a valid APT-mapping from $\mathcal{Q}$ to some derivation tree $\mathcal{T}$ having $\mathfrak{T}$ as a prefix. We can conclude thanks to Property 23.

($\Rightarrow$) 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 with derivation tree $\mathcal{T} = DT(S)$. As in the construction given before Definition 34, we can chose some mapping $\pi^{\mathrm{a}}_{\mathcal{T}}$ of the atoms of Q, and build from this mapping an APT $\mathcal{Q}$ of Q. Now, in this particular π, the root of $\mathcal{Q}$ 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 $\mathfrak{T}_b$, there exists another homomorphism π' from Q to some F'_k obtained by means of a derivation S'. Let us recompute an APT $\mathcal{Q}'$ (the same result might be obtained). Then (see proof of Property 23), there is a valid APT mapping $\Gamma = (\Pi, \pi_1, \ldots, \pi_k)$ from $\mathcal{Q}'$ to $DT(S')$, since $DT(S)$ is a prefix tree of some $\mathfrak{T}$ generated from $\mathfrak{T}_b$. Γ is thus a valid APT mapping from $\mathcal{Q}$ to $\mathfrak{T}$.

Now let us define the mapping Ξ as follows: if Q^{at} is the root of $\mathcal{Q}$, then $\Xi(Q^{\mathrm{at}}) = \Pi(Q^{\mathrm{at}})$, otherwise $\Xi(Q^{\mathrm{at}}) = f(\Pi(Q^{\mathrm{at}}))$. For each term t in the atom term bag Q^{at}_i of $\mathcal{Q}$ such that $\Xi(Q^{\mathrm{at}}_i) = B_j$, we define $\pi'_i(t) = \psi_{B_j \to f(B_j)} \circ \pi_i$. Let us consider the APT mapping $\Gamma' = (\Xi, \pi'_1, \ldots, \pi'_k)$ from $\mathcal{Q}$ to $\mathfrak{T}_b$. It is immediate to check that Γ' is valid.

□

4.5.3 A bounded validation for APT-mappings

Although Property 24 brings us closer to our goal to obtain an algorithm for *gbts* deduction, there is still the need to guess the generated tree $(\mathfrak{T}, f)$ used to validate an APT-mapping (Definition 36). 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 we establish an upper bound for each validation step (*i.e.*, if we have validated an initial segment $\mathcal{Q}'$ of the APT $\mathcal{Q}$ of Q, and $Q^{\mathrm{at}'}$ is a child of some bag Q^{at} in $\mathcal{Q}'$, how do we validate $Q' \cup \{Q^{\mathrm{at}'}\}$?).

Property 25. *Let $\mathcal{Q}$ be an APT of Q, and Γ be a valid APT-mapping from $\mathcal{Q}$ to a blocked tree $\mathfrak{T}_b$. Let $\mathcal{Q}'$ be a prefix tree of $\mathcal{Q}$, and Γ' be the restriction of Γ to $\mathcal{Q}'$. Note that Γ' is a valid APT-mapping from $\mathcal{Q}'$ to $\mathfrak{T}_b$.*

Consider now any proof $((\mathfrak{T}', f'), \Xi')$ of Γ' (see Definition 36).[9] *Then, there exists a proof $((\mathfrak{T}'', f''), \Xi'')$ of Γ such that $((\mathfrak{T}', f'), \Xi')$ is a subproof of $((\mathfrak{T}'', f''), \Xi'')$.*

Proof. Let us consider a proof $((\mathfrak{T}', f'), \Xi')$ of Γ'. As shown in the proof of Property 24, this proof corresponds to a homomorphism π' of Q' to $(\mathfrak{T}', f')$. Now consider any leaf bag $Q^{\mathrm{at}'}$ of $\mathcal{Q}'$, that is the root of a tree in $\mathcal{Q}$. Γ and Γ' can map $Q^{\mathrm{at}'}$ to different bags in $(\mathfrak{T}', f')$ and $(\mathfrak{T}, f)$. However, these bags are equivalent (according to Definition 18) to the same bag in $\mathfrak{T}_b$. 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 $Q^{\mathrm{at}'}$ can be mapped in the same way under the bag of $(\mathfrak{T}', f')$. This construction leads to a homomorphism π'' from Q to some $(\mathfrak{T}'', f'')$ that extends π'. Using again the proof of Property 24, this homomorphism can be used to build a proof $((\mathfrak{T}'', f''), \Xi'')$ of Γ, that is a superproof of $((\mathfrak{T}', f'), \Xi')$. □

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

9. Note that this proof $((\mathfrak{T}', f'), \Xi')$ is not necessarily a subproof of the existing proof $((\mathfrak{T}, f), \Xi)$ of Γ (*i.e.*, $(\mathfrak{T}', f')$ is not necessarily an initial segment of $(\mathfrak{T}, f)$ and Ξ' is not necessarily the restriction of Ξ).

Algorithm 2: ValidateAPT

Data: A blocked tree $\mathfrak{T}_b$, an APT $\mathcal{Q}$, and an APT-mapping Γ from $\mathcal{Q}$ to $\mathfrak{T}_b$.
Result: YES if Γ is valid, NO otherwise.

```
Explored := ∅;
for i = 1 to |Q| do
    Q^at := some bag of Q s.t. either (parent(Q^at), −) ∈ Explored, or Q^at is the root
    of Q;
    if joins(Γ, Q^at) ≠ ∅ then
        Explored := Explored ∪{(Q^at, joins(Γ, Q^at))};
    else
        return NO;
return YES;
```

It remains now to explain the procedure *joins* that checks whether a valid APT-mapping of a subtree $\mathcal{Q}'$ of $\mathcal{Q}$ can be extended to a child $Q^{\text{at}} \in \mathcal{Q} \setminus \mathcal{Q}'$ of some bag of $\mathcal{Q}'$. Let us consider a proof $((\mathfrak{T}', f'), \Xi')$ of $\Gamma = (\Pi, \pi_1, \ldots, \pi_k)$ being a valid APT-mapping of $\mathcal{Q}'$. According to Def. 36 and Prop. 24, it is sufficient to:

- find a bag B_n that can be obtained by a *bag copy sequence* $B_1, \ldots, B_n$ where $B_1 = \Xi'(\text{parent}(Q^{\text{at}}))$, B_n is a bag equivalent to $\Pi(Q^{\text{at}})$, and for $1 < i \leq n$, B_i is obtained by a bag copy (see Definition 20) under B_{i-1}. Since $\Pi(Q^{\text{at}})$ and B_n are equivalent, there is a bijection from the terms of $\Pi(Q^{\text{at}})$ to the terms of B_n, that we denote by ψ.

- it remains now to check that for every term t appearing in an atom of Q^{at}, t is a term that belongs to a bag $Q^{\text{at}\prime}$ in the branch from the root of $\mathcal{Q}$ to Q^{at}, and $\Xi'(t) = \psi(\pi(t))$ (where π is the mapping defined in the APT-mapping from the terms of Q^{at} to those of $\Pi(Q^{\text{at}})$). In that case, the call to *joins* returns $\psi \circ \pi$, ensuring that we are able to evaluate the joins of the next bags.

We last prove that there exists a "short" proof of every valid APT-mapping.

Property 26. *Let $\mathcal{Q}$ be an APT of Q, and Γ be a valid APT-mapping from $\mathcal{Q}$ to a blocked tree $\mathfrak{T}_b$. There exists a proof $((\mathfrak{T}, f), \Xi)$ of Γ such that for any two bags Q_i^{at} and Q_j^{at} from $\mathcal{Q}$ where Q_i^{at} is a child of Q_j^{at}, the distance between $\Xi(Q\text{at}_i)$ and $\Xi(Q\text{at}_j)$ in $\mathfrak{T}$ is at most $p \times f^f$, where p is the number of abstract patterns, and f the maximum size of a rule frontier.*

Proof. We prove the result by induction on the number of atom-term bags in $\mathcal{Q}$. If $\mathcal{Q}$ has only one atom-term bag, the property is trivially true. Let us assume the result to be true for any APT-mapping whose APT is of size n, and let $\mathcal{Q}$ be a APT of size $n+1$, and Γ be a valid APT-mapping of $\mathcal{Q}$. Let Q_c^{at} be an arbitrary leaf of $\mathcal{Q}$, let $\mathcal{Q}'$ be equal to $\mathcal{Q} \setminus \{Q_c\}$, and let Γ' be the restriction of Γ to $\mathcal{Q}'$. By induction assumption, there exists a proof $((\mathfrak{T}', f'), \Xi')$ of Γ' fulfilling the claimed property. By the proof of Property 25, this proof can be extended to a proof $((\mathfrak{T}, f), \Xi)$, such that Q_c^{at} is mapped to a descendant of $\Xi(Q_p^{\text{at}})$, where Q_p^{at} is the parent of Q_c^{at} in $\mathcal{Q}$. Moreover, the provided construction allows to ensure

that there are no two distinct bags B_i and B_j on the path from $\Xi(Q_p^{\text{at}})$ to $\Xi(Q_c^{\text{at}})$ fulfilling the following two conditions:

- $f(B_i) = f(B_j)$;
- $\forall x \in X_s, \psi_{B_i \to f(B_i)}(x) = \psi_{B_j \to f(B_j)}(x)$, where X_s is the image of the set of variables that appear both in Q_c^{at} and $\mathcal{Q}'$.

Then the distance between $\Xi(Q_p^{\text{at}})$ and $\Xi(Q_c^{\text{at}})$ has to be less than $p \times f^f$. Indeed, X_s should belong to the frontier of $\Xi(Q_c^{\text{at}})$, by the running intersection property of $\mathfrak{T}$. There is at most f^f ways of arranging these terms, thus providing the claimed upper-bound. □

4.6 Complexity Analysis and Worst-Case Optimal Adaptation to Subclasses

We provide a worst-case complexity analysis of the proposed algorithm, and present some small modifications that can be adopted in order to make the algorithm worst-case optimal for relevant subclasses of rules. Table 3 provides a summary of the notation used for the complexity analysis.

Table 3: Notations used in the complexity analysis

Notation	Signification
b	upper-bound on the number of terms in any abstract pattern
q	number of atoms plus number of terms in the query
f	maximum size of a rule frontier
a_B	maximum number of atoms in a rule body
a_H	maximum number of atoms in a rule head
t_B	maximum number of terms in a rule body
t_H	maximum number of terms in a rule head
$\lvert\mathcal{R}\rvert$	number of rules
p	number of abstract patterns
w	maximum arity of a predicate
s	number of predicates appearing in the rule set

4.6.1 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 $\mathfrak{T}_b$ of $(F, \mathcal{R})$ (note that this is done independently of the query)
- for every APT $\mathcal{Q}$ of Q, for every APT-mapping Γ of $\mathcal{Q}$ to $\mathfrak{T}_b$, if ValidateAPT returns yes, return yes (and return no at the end otherwise).

The first step is done linearly in the number of evolution and creation rules. 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 22.

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 $N_Q = \mathcal{O}(q^q)$ (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 is $\mathcal{O}(pb^f)$ and thus $N_\Gamma = \mathcal{O}(p^q \times b^{fq})$;
- 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 $\mathcal{O}(p \times f^f)$ bags (see Property 26).

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

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 27. *CQ entailment for $\mathcal{gbts}$ 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.

4.6.2 Adaptation to relevant subclasses

We now show how to adapt the algorithm to the subclasses of $\mathcal{gbts}$ that have smaller worst-case complexities. 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 three cases:

- (weakly) guarded rules, whose combined complexity in the bounded arity case drops to ExpTime,
- guarded frontier-one rules, whose combined complexity in the unbounded 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.

If we consider only guarded frontier-one rules, the number of possible homomorphisms decreases. Indeed, the set of atoms whose terms are included in a given bag is upper-bounded by $a_H + s.t_H$: the atoms that have been created at the creation of the bag, plus atoms that may be created afterwards. However, these atoms must be of the from $r(x, \ldots, x)$, since the considered rules are frontier-one, hence at most $s.t_H$. This results in a simply exponential number of patterns, which provides the claimed upper-bound.

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 bag), 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.

5. Matching Lower Complexity Bounds for *gbts* Subclasses

We now provide the missing hardness results to fully substantiate all complexity results displayed in Figure 1 and Table 1.

5.1 Data Complexity of Guarded Frontier-One Rules is PTime-hard

PTime hardness for *gfr1* rules is not hard to establish and follows from known results (for instance, the description logic $\mathcal{EL}$ is subsumed by *gfr1* rules and known to have PTime-hard data complexity). For the sake of self-containedness, we will give a direct reduction from one of the prototypical PTime problems: entailment in propositional Horn logic.

Theorem 28. *CQ entailment under constant-free gfr1 rules is* PTime*-hard for data complexity.*

Proof. Given a set $\mathcal{H}$ of propositional Horn clauses, we introduce for every propositional atom a occurring therein a constant c_a. We also introduce one additional constant *nil*. Moreover, for every Horn clause $C \in \mathcal{H}$ with $C = a_1 \wedge \ldots \wedge a_n \rightarrow a$, we introduce constants $b_{C,1}, \ldots, b_{C,n}$ and let F consist of $\mathit{entails}(b_{C,1}, c_a)$, $\mathit{first}(b_{C,i}, c_{a_i})$ for all $i \in \{1, \ldots, n\}$, as

well as $\mathit{rest}(b_{C,n}, \mathit{nil})$, and $\mathit{rest}(b_{C,i}, b_{C,i+1})$ for all $i \in \{1, \ldots, n-1\}$, also let F contain $\mathit{entailed}(\mathit{nil})$. Then the propositional atom a is entailed by $\mathcal{H}$ exactly if $F, \mathcal{R} \models Q$ with $Q = \mathit{entailed}(c_a)$ and $\mathcal{R}$ containing the rules:

$$\begin{aligned} \mathit{first}(y,z) \wedge \mathit{entailed}(z) \wedge \mathit{rest}(y,z') \wedge \mathit{entailed}(z') &\rightarrow \mathit{entailed}(y), \\ \mathit{entailed}(y) \wedge \mathit{entails}(y,z) &\rightarrow \mathit{entailed}(z). \end{aligned}$$

□

5.2 Combined Complexity of Guarded Frontier-One Rules is ExpTime-hard

We prove ExpTime-hardness of CQ entailment under *gfr1* rules by showing that any standard reasoning task in the description logic Role-Bounded Horn-$\mathcal{ALC}$, for which ExpTime-hardness is known (Krötzsch, Rudolph, & Hitzler, 2007, 2013), can be polynomially reduced to the considered problem.

We start by defining this problem. In order to avoid syntactic overload, we will stick to first-order logic syntax and refrain from using the traditional description-logic-style notation.

Definition 37 (Role-Bounded Horn-$\mathcal{ALC}$)**.** *Let* Pr_1 *an infinite set of unary predicates and let* Pr_2 *be a finite set of binary predicates. A* reduced normalized Horn-$\mathcal{ALC}[\mathsf{Pr}_2]$ terminology $\mathcal{T}$ *is a set of rules having one of the following shapes (with* $p_1, p_2, p_3 \in \mathsf{Pr}_1$ *and* $r \in \mathsf{Pr}_2$*):*

(A) $p_1(x) \rightarrow p_2(x)$

(B) $p_1(x) \wedge p_2(x) \rightarrow p_3(x)$

(C) $r(x,y) \wedge p_1(y) \rightarrow p_2(x)$

(D) $p_1(x) \wedge r(x,y) \rightarrow p_2(y)$

(E) $p_1(x) \rightarrow \exists y.(r(x,y) \wedge p_2(y))$

We refer to the problem of deciding if for some $\mathcal{T}$ *and* $p_1, p_2 \in \mathsf{Pr}_1$ *holds* $\mathcal{T} \models \forall x(p_1(x) \rightarrow p_2(x))$ *as* unary subsumption checking*.*

Theorem 29 ((Krötzsch et al., 2013), Section 6.2.)**.** *There is a finite set* Pr_2 *such that unary subsumption checking for reduced normalized Horn-*$\mathcal{ALC}[\mathsf{Pr}_2]$ *terminologies is* ExpTime*-hard.*

The following corollary is now a straightforward consequence.

Corollary 30. *CQ entailment under constant-free* ***gfr1*** *rules is* ExpTime*-hard for combined complexity.*

Proof. Clearly, given a reduced normalized Horn-$\mathcal{ALC}[\mathsf{Pr}_2]$ terminology $\mathcal{T}$ and $p_1, p_2 \in \mathsf{Pr}_1$, the unary subsumption entailment $\mathcal{T} \models \forall x(p_1(x) \rightarrow p_2(x))$ is to be confirmed if and only if $F, \mathcal{R} \models Q$ with $F = \{p_1(a)\}$, $\mathcal{R} = \mathcal{T}$ and $Q = p_2(a)$. The latter can be conceived as a CQ entailment problem of the desired type, since $\mathcal{T}$ is a *gfr1* rule set. □

Note that our line of argumentation actually does not require the set Pr_2 to be fixed, however, this will be a necessary precondition in the next section where we use the same logic, hence we have introduced the logic in this form from the beginning.

5.3 Data Complexity of Weakly Guarded Frontier-One Rules is ExpTime-hard

We will now show that the data complexity for deciding CQ entailment under *wgfr1* rules is ExpTime-hard. Again, we obtain the result by showing that a Horn-$\mathcal{ALC}[\mathsf{Pr}_2]$ reasoning problem can be reduced to CQ entailment under *wgfr1* rules but this time even with fixed rule set and query.

Definition 38. *Given a set Pr_2, let $\mathcal{R}_{\text{fix}}$ be the fixed wgfr1 rule set containing the following rules (where r ranges over all elements of Pr_2):*

1. $typeA(z_1, z_2) \wedge in(x, z_1) \rightarrow in(x, z_2)$
2. $typeB(z_1, z_2, z_3) \wedge in(x, z_1) \wedge in(x, z_2) \rightarrow in(x, z_3)$
3. $typeC_r(z_1, z_2) \wedge r(x, y) \wedge in(y, z_1) \rightarrow in(x, z_2)$
4. $typeD_r(z_1, z_2) \wedge in(x, z_1) \wedge r(x, y) \rightarrow in(y, z_2)$
5. $typeE_r(z_1, z_2) \wedge in(x, z_1) \rightarrow r(x, y) \wedge in(y, z_2)$
6. $test(x) \wedge sub(z) \rightarrow in(x, z)$
7. $test(x) \wedge in(x, z) \wedge super(z) \rightarrow match$

Given a reduced normalized Horn-$\mathcal{ALC}[\mathsf{Pr}_2]$ terminology $\mathcal{T}$, we let $F_{\mathcal{T}}$ be the fact containing

1. $typeA(c_{p_1}, c_{p_2})$ *for any* $R = p_1(x) \rightarrow p_2(x)$ *from* $\mathcal{T}$,
2. $typeB(c_{p_1}, c_{p_2}, c_{p_3})$ *for any* $R = p_1(x) \wedge p_2(x) \rightarrow p_3(x)$ *from* $\mathcal{T}$,
3. $typeC_r(c_{p_1}, c_{p_2})$ *for any* $R = r(x, y) \wedge p_1(y) \rightarrow p_2(x)$ *from* $\mathcal{T}$,
4. $typeD_r(c_{p_1}, c_{p_2})$ *for any* $R = p_1(x) \wedge r(x, y) \rightarrow p_2(y)$ *from* $\mathcal{T}$, *and*
5. $typeE_r(c_{p_1}, c_{p_2})$ *for any* $R = p_1(x) \rightarrow \exists y.(r(x, y) \wedge p_2(y))$ *from* $\mathcal{T}$.

Also, for $p_1, p_2 \in \mathsf{Pr}_1$, we let $F_{p_1 p_2} = \{test(a), sub(c_{p_1}), super(c_{p_2})\}$

The following property now lists two straightforward observations.

Property 31. *$\mathcal{R}_{\text{fix}}$ is a wgfr1 rule set. $F_{\mathcal{T}}$ can be computed in polynomial time and is of polynomial size with respect to $\mathcal{T}$.*

The next lemma establishes that subsumption in Horn-$\mathcal{ALC}[\mathsf{Pr}_2]$ can be reduced to *CQ* entailment w.r.t. a fixed *wgfr1* rule set.

Lemma 32. *Let $\mathcal{T}$ be a reduced normalized Horn-$\mathcal{ALC}[\mathsf{Pr}_2]$ terminology and let $p_1, p_2 \in \mathsf{Pr}_1$. Then $\mathcal{T} \models \forall x(p_1(x) \rightarrow p_2(x))$ if and only if $F_{\mathcal{T}} \cup F_{p_1 p_2}, \mathcal{R}_{\text{fix}} \models match$.*

Proof. We successively prove both directions of the equivalence.

⇒ Assume to the contrary that $F_{\mathcal{T}} \cup F_{p_1p_2}, \mathcal{R}_{\text{fix}} \models \mathit{match}$ does not hold, thus we find a model of $F_{\mathcal{T}} \cup F_{p_1p_2}, \mathcal{R}_{\text{fix}}$ where *match* is false. We then can use this model to construct a model of $\mathcal{T}$ not satisfying $\forall x(p_1(x) \to p_2(x))$ using the following defining equation for every $p_i \in \mathsf{Pr}_1$:

$$\forall x(p_i(x) \leftrightarrow \mathit{in}(x, c_{p_i})).$$

It can be readily checked that this model indeed satisfies $\mathcal{T}$. Moreover it satisfies $p_1(a)$ but not $p_2(a)$, therefore $\forall x(p_1(x) \to p_2(x))$ does not hold.

⇐ Assume to the contrary that $\mathcal{T} \models \forall x(p_1(x) \to p_2(x))$ does not hold, i.e. we find a model of $\mathcal{T}$ not satisfying $\forall x(p_1(x) \to p_2(x))$ (i.e., in this model there must be one element e in the extension of p_1 but not of p_2). We now use this model to define a model of $F_{\mathcal{T}} \cup F_{p_1p_2}, \mathcal{R}_{\text{fix}}$ by adding to the domain a new element c_{p_i} for every $p_i \in \mathsf{Pr}_1$ and let the interpretation function on the eponymous constants be the identity. The interpretation of the predicates $\mathit{typeA}, \ldots, \mathit{typeE}$ is as explicitly stated in $F_{\mathcal{T}}$, the interpretation of the *in* predicate is defined as the minimal set such that

$$\forall x(p_i(x) \leftrightarrow \mathit{in}(x, c_{p_i}))$$

holds for all $p_i \in \mathsf{Pr}_1$. Finally, we let *sub* hold for c_{p_1}, we let *super* hold for c_{p_2}, and we let *test* hold for a, where the constant a is mapped to the domain element e mentioned above. Then it can be easily checked that the obtained model satisfies all of $F_{\mathcal{T}} \cup F_{p_1p_2}, \mathcal{R}_{\text{fix}}$ but not *match*.

□

Theorem 33 (Data Complexity of *wgfr1* Rules)**.** BCQ-Entailment *under constant-free wgfr1 rules is* ExpTime*-hard for data complexity.*

Proof. By Theorem 29, unary subsumption checking for reduced normalized Horn-$\mathcal{ALC}[\mathsf{Pr}_2]$ terminologies is ExpTime-hard. By Lemma 32 and thanks to Proposition 31, any such problem can be polynomially reduced to a CQ entailment problem $F, \mathcal{R} \models q$ for uniform $\mathcal{R}$ and q. Consequently, the data complexity of conjunctive query entailment under *wgfr1* rules must be ExpTime-hard as well. □

5.4 Combined Complexity of Frontier-One Rules is 2ExpTime-hard

In this section, we show that *fr1* rules are 2ExpTime-hard for combined complexity no matter whether predicate arity is bounded or not. Our proof reuses the general strategy and many technical tricks from a construction used to show 2ExpTime-hardness for CQ entailment in the DL $\mathcal{ALCI}$ from (Lutz, 2007). Still, many adaptations were done in order to make the construction fit our language fragment and to simplify unnecessarily complicated parts.

We prove the desired result via a reduction of CQ entailment w.r.t. *fr1* rules to the word problem of an alternating Turing machine with exponential space, which will be formally introduced next.

Definition 39. *An* alternating Turing machine $\mathcal{M}$*, short ATM, is a quadruple* $(\mathfrak{Q}, \Gamma, q_0, \Delta)$ *where:*

- $\mathfrak{Q}$ *is the set of* states[10]*, which can be partitioned into existential states* $\mathfrak{Q}_\exists$ *and universal states* $\mathfrak{Q}_\forall$*.*
- Γ *is a finite* alphabet*, containing the* blank symbol $\square$*,*
- $q_0 \in \mathfrak{Q}$ *is the* initial state*,*
- $\Delta \subseteq \mathfrak{Q} \times \Sigma \times \mathfrak{Q} \times \Sigma \times \{L, R\}$ *is the* transition relation*.*

A configuration *of an ATM is a word* wqw' *where* $w, w' \in \Gamma^*$ *and* $q \in \mathfrak{Q}$*. The* successor configurations $\Delta(wqw')$ *of a configuration* wqw' *are defined as:*

$$\begin{aligned}\Delta(wqw') = \quad & \{vq'\gamma''\gamma'v' \mid w = v\gamma'', w' = \gamma v', (\gamma, q, \gamma', q', L) \in \Delta\} \\ & \cup \{w\gamma'q'v' \mid w' = \gamma v', (\gamma, q, \gamma', q', R) \in \Delta\} \\ & \cup \{w\gamma'q' \mid w' = \epsilon, (\square, q, \gamma', q', R) \in \Delta\}.\end{aligned}$$

The set of indirect successors *of a configuration* wqw' *is the smallest set of configurations that contains* wqw' *and that is closed under the successor relation.*

A halting configuration *is of the form* wqw' *with* $\Delta(wqw') = \emptyset$*. The set of* accepting configurations *is the smallest set of configurations such that:*

- wqw' *is accepting if there exists* $vq'v' \in \Delta(wqw')$ *is accepting in case of* $q \in \mathfrak{Q}_\exists$*,*
- wqw' *is accepting if all* $vq'v' \in \Delta(wqw')$ *are accepting in case of* $q \in \mathfrak{Q}_\forall$*.*

An ATM is said to accept a word $\mathfrak{w} \in \Gamma^*$*, if* $q_0 w$ *is accepting.*

An ATM is exponentially space bounded *if for any* $\mathfrak{w} \in \Gamma^*$*, every indirect successor* vqv' *of* $q_0\mathfrak{w}$ *satisfies that* $|vv'| < 2^{|\mathfrak{w}|}$*.*

According to (Chandra, Kozen, & Stockmeyer, 1981b), there is an exponentially space bounded ATM $\mathcal{M}^*$, whose word problem is 2ExpTime-hard. In order to simplify our argument, we will, however, not directly simulate a run of this Turing machine on a word $\mathfrak{w}$. Rather, given $\mathcal{M}^*$ and a word $\mathfrak{w}$ it is straightforward to construct an ATM $\mathcal{M}^*_\mathfrak{w}$ such that $\mathcal{M}^*$ accepts $\mathfrak{w}$ exactly if $\mathcal{M}^*_\mathfrak{w}$ accepts the empty word ϵ.[11] Clearly, the size of $\mathcal{M}^*_\mathfrak{w}$ is polynomially bounded by $n = |\mathfrak{w}|$.

In the following, we will thus show how, given an exponentially space bounded ATM $\mathcal{M}$ and a word $\mathfrak{w}$, we can construct a fact F, rule set $\mathcal{R}$ and query Q – the size of all being polynomially bounded by n – such that $F, \mathcal{R} \models q$ iff $\mathcal{M}^*_\mathfrak{w} = (\mathfrak{Q}, \Gamma, q_0, \Delta)$ accepts the empty word. Thereby, the minimal model of F and $\mathcal{R}$ will contain elements representing the initial and all its (indirect) successor configurations. These configurations will themselves be endowed with a tree structure that stores the content of the exponentially bounded

10. $\mathfrak{Q}$ (with possible subscripts) is used for state sets in order to avoid a notational clash with conjunctive queries denoted by Q.
11. Such a machine can be easily obtained: add new states and transitions that first write $\mathfrak{w}$ to the tape, second go back to the starting position, and third switch to the initial state of the original Turing machine.

memory. The most intricate task to be solved will be to model memory preservation from one configuration to its successors.

We start by introducing some predicates and their intuitive meaning:

- *conf*: unary predicate to distinguish elements representing configurations from other elements;
- *firstconf*: unary predicate to denote the initial configuration;
- $trans_\delta$ with $\delta \in \Delta$: set of binary predicates. $trans_\delta$ connects a configuration with its successor configuration that was introduced due to δ;
- $state_q$ with $q \in \mathfrak{Q}$: set of unary predicates indicating the state of the configuration wqw';
- $symbol_\gamma$ with $\gamma \in \Gamma$: set of unary predicates indicating the symbol at the heads current position, i.e. $symbol_\gamma$ holds for the configuration $wq\gamma w'$;
- *accepting*: unary predicate indicating if a configuration is accepting;
- *wire* binary predicate used later for memory operations;
- *fw* unary predicate used later for memory operations.

We are now ready to provide first constituents of the fact F and of the rule set $\mathcal{R}$. We let F contain the facts:

$$\{conf(init), firstconf(init), state_{q_0}(init)\}. \tag{1}$$

For every $\delta = (q, \gamma, q', \gamma', D) \in \Delta$ (with $D \in \{L, R\}$), let $\mathcal{R}$ contain:

$$state_q(x) \wedge symbol_\gamma(x) \rightarrow trans_\delta(x, y) \wedge wire(x, u) \wedge fw(u) \wedge wire(u, v) \wedge fw(v) \wedge wire(v, y) \tag{2}$$

$$trans_\delta(x, y) \rightarrow conf(y) \wedge state_{q'}(y) \tag{3}$$

Clearly, by means of these rules, we create the successor configurations y reached from a transition predicate from given configuration x. The additionally introduced sequence "$\xrightarrow{wire} u \xrightarrow{wire} v \xrightarrow{wire}$" between x and y will come handy later for memory preservation purposes. Figure 11 displays the structure of the configuration tree thus constructed.

Next we take care of the implementation of the acceptance condition for configurations. For every $q \in \mathfrak{Q}_\exists$ and every $\delta = (q, \gamma, q', \gamma', D) \in \Delta$, we add the rule:

$$trans_\delta(x, y) \wedge accept(y) \rightarrow accept(x). \tag{4}$$

For every $q \in \mathfrak{Q}_\forall$ and $\gamma \in \Gamma$, we add the rule:

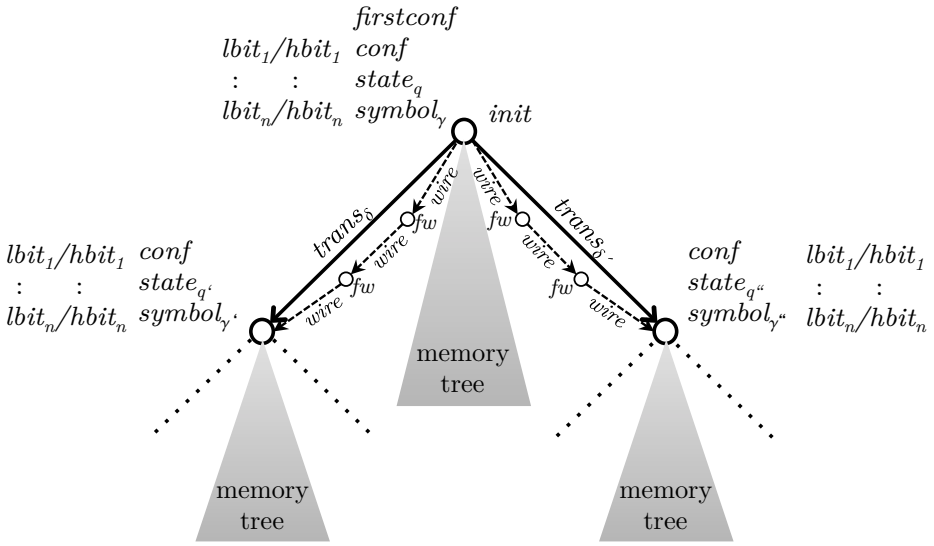

Figure 11: Structure of the configuration tree in the constructed model.

$$\bigwedge_{\delta=(q,\gamma,q',\gamma',D)\in\Delta} trans_\delta(x, y_\delta) \wedge accept(y_\delta) \rightarrow accept(x). \tag{5}$$

This way, as required, acceptance is propagated backward from successors to predecessors.

Consequently, the query to be posed against the "computation structure" described by our rule set should ask if the initial configuration is accepting, i.e.:

$$Q = accept(init). \tag{6}$$

Next, we prepare the implementation of the memory access. To this end, we encode the position of the head (that is, the length of the word w) in a configuration wqw' as an n-digit binary number (note that this allows us to address 2^n positions which is sufficient for the required exponential memory). We will use unary predicates $hbit_k$, $lbit_k$ with $1 \leq k \leq n$ for the following purpose: $hbit_k$ holds for an element representing a configuration wqw', if the kth bit of the configuration's head position (i.e. the number $|w|$) expressed in binary format is 1. If the bit is 0, then $lbit_k$ holds instead.

Clearly, the initial position of the head is 0 (as we start from configuration $q_0\mathfrak{w}$), thus for the initial configuration (represented by $init$) all bits must be 0. Hence we let $F_{\mathcal{M},\mathfrak{w}}$ contain $lbit_k(init)$ for all $1 \leq k \leq n$.

In the course of a state transition $\delta = (q, \gamma, q', \gamma', D) \in \Delta$, the head's position may be increased by one (in case $D = R$) or decreased by 1 (in case $D = L$). The next rules implement this behavior, hence given a configuration's head position, they effectively compute the head position of this configuration's direct successors. For every $\delta = (q, \gamma, q', \gamma', D) \in \Delta$

with $D = R$ we let $\mathcal{R}_{\mathcal{M},\mathfrak{w}}$ contain the rules (where k ranges from 1 to n and m ranges from 1 to k):

$$trans_\delta(x,y) \wedge \bigwedge_{l \leq k} hbit_l(x) \quad \rightarrow lbit_k(y) \tag{7}$$

$$trans_\delta(x,y) \wedge lbit_k(x) \wedge \bigwedge_{l < k} hbit_l(x) \quad \rightarrow hbit_k(y) \tag{8}$$

$$trans_\delta(x,y) \wedge lbit_k(x) \wedge lbit_m(x) \quad \rightarrow lbit_k(y) \tag{9}$$

$$trans_\delta(x,y) \wedge hbit_k(x) \wedge lbit_m(x) \quad \rightarrow hbit_k(y) \tag{10}$$

and for every $\delta = (q, \gamma, q', \gamma', D) \in \Delta$ with $D = L$ we let $\mathcal{R}_{\mathcal{M},\mathfrak{w}}$ contain the rules (ranges of k and m as above):

$$trans_\delta(x,y) \wedge \bigwedge_{l \leq k} lbit_l(x) \quad \rightarrow hbit_k(y) \tag{11}$$

$$trans_\delta(x,y) \wedge hbit_k(x) \wedge \bigwedge_{l < k} lbit_l(x) \quad \rightarrow lbit_k(y) \tag{12}$$

$$trans_\delta(x,y) \wedge hbit_k(x) \wedge hbit_m(x) \quad \rightarrow hbit_k(y) \tag{13}$$

$$trans_\delta(x,y) \wedge lbit_k(x) \wedge hbit_m(x) \quad \rightarrow lbit_k(y) \tag{14}$$

In the next steps, we need to implement the exponential size memory of our Turing machine. At the same time, the memory should be "accessible" by polynomial size rule bodies. Thus we organize the memory as a binary tree of polynomial depth having exponentially (that is 2^n) many leaves. Thus, for every configuration element, we create a tree of depth n having the configuration element as root and where the configuration's tape content is stored in the leaves. We use the following vocabulary:

- $level_k$ with $0 \leq k \leq n$: set of unary predicates stating for each node inside the memory tree its depth.
- *leftchild*, *rightchild*: the two (binary) child predicates of the memory tree.
- *child*: a (binary) predicate subsuming the two above.
- $entry_\gamma$ with $\gamma \in \Gamma$: set of unary predicates indicating for every leaf of the memory tree the symbol stored there.

We now create the memory tree level by level (with k ranging from 1 to n):

$$conf(x) \rightarrow level_0(x) \tag{15}$$

$$level_{k-1}(x) \rightarrow leftchild(x,y) \wedge child(x,y) \wedge wired(x,y) \wedge wired(y,x) \wedge level_k(y) \tag{16}$$

$$level_{k-1}(x) \rightarrow rightchild(x,y) \wedge child(x,y) \wedge wired(x,y) \wedge wired(y,x) \wedge level_k(y) \tag{17}$$

$$\tag{18}$$

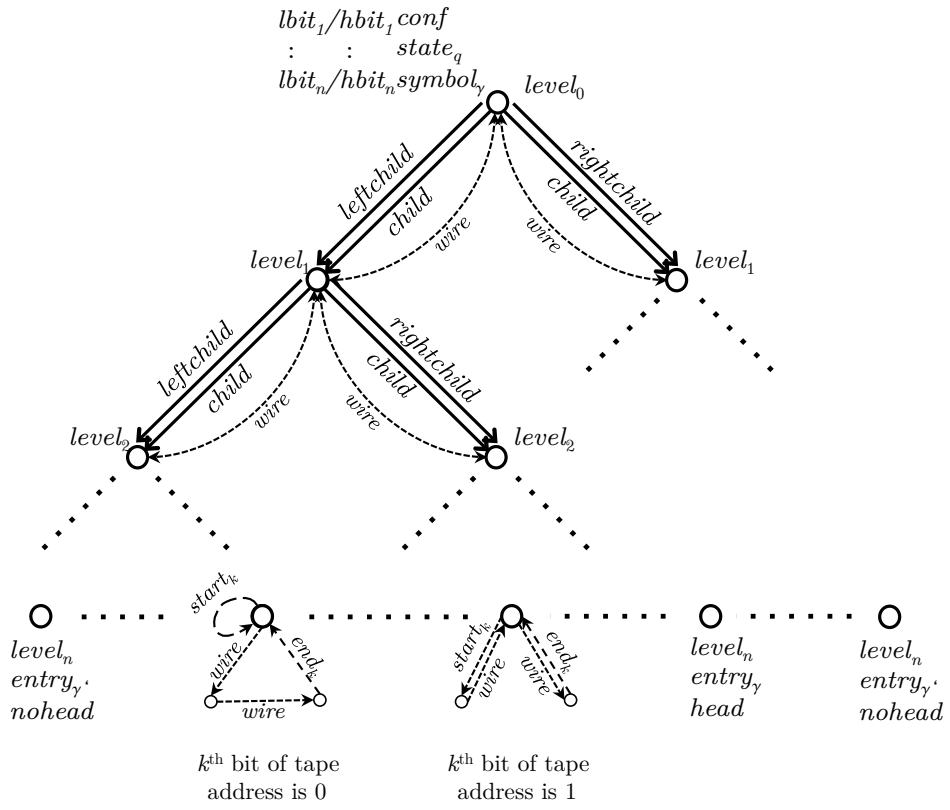

Figure 12: Structure of the memory tree in the constructed model.

The leaf nodes of the memory tree thus created (i.e., the elements satisfying $level_n$) will be made to carry two types of information: (a) the current symbol stored in the corresponding tape cell and (b) the tape cell's "address" in binary encoding. The latter will be realized as follows: if the kth bit of the binary representation of the address is clear, the leaf node ν will be extended by a structure containing two newly introduced elements ν_1 and ν_2 which will be connected with ν via the following binary predicates: $start_k(\nu, \nu)$, $wired(\nu, \nu_1)$, $wired(\nu_1, \nu_2)$, and $end_k(\nu_2, \nu)$. In case the kth bit is set, we will also introduce new elements ν_1 and ν_2 but they will be connected with ν in a different way, namely: $start_k(\nu, \nu_1)$, $wired(\nu_1, \nu)$, $wired(\nu, \nu_2)$, and $end_k(\nu_2, \nu)$. The reason for this peculiar way of encoding the address information will become apparent in the sequel. Figure 12 depicts the structure of the memory tree constructed under each configuration tree.

The following rules realize the aforementioned address representation, exploiting the fact that the kth address bit will be 0 if the considered leaf node's ancestor on level $k-1$ is connected with the ancestor on level k via *leftchild* and it will be 0 if the connection is via *rightchild*. Hence we let $\mathcal{R}_{\mathcal{M},\mathfrak{w}}$ contain the rules (with k ranging from 1 to n as above):

$$leftchild(x_{k-1}, x_k) \wedge \bigwedge_{i=k+1}^{n}(child(x_{i-1}, x_i)) \wedge level_n(x_n) \rightarrow start_k(x_n, x_n) \wedge wired(x_n, x'_n) \wedge wired(x'_n, x''_n) \wedge end_k(x''_n, x_n) \quad (19)$$

$$rightchild(x_{k-1}, x_k) \wedge \bigwedge_{i=k+1}^{n}(child(x_{i-1}, x_i)) \wedge level_n(x_n) \rightarrow start_k(x_n, x'_n) \wedge wired(x'_n, x_n) \wedge wired(x_n, x''_n) \wedge end_k(x''_n, x_n) \quad (20)$$

One of the purposes of the previous construction is to mark in each memory tree the leaf corresponding to the current head position by a unary predicate *head* and all other leaves by another unary predicate *nohead*. To this end, we encode the head position stored in the configuration elements via the predicates $lbit_k$ and $hbit_k$ in a "structural way", similar to our encoding in the leaves:

$$lbit_k(x) \rightarrow rootstart_k(x, x) \quad (21)$$

$$hbit_k(x) \rightarrow rootstart_k(x, x') \wedge wired(x', x) \quad (22)$$

For the assignment of *head* and *nohead* to leaf nodes, we now exploit two facts. First, the kth bit of the head address – stored in a configuration element ν_c – and the kth bit of the address of a leaf node ν_l of the same configuration element coincide, if there are nodes $\nu_1, \ldots, \nu_{n-1}$ such that there is a path

$$\nu_c \xrightarrow{rootstart_k} \nu_1 \xrightarrow{wired} \ldots \xrightarrow{wired} \nu_{n-1} \xrightarrow{end_k} \nu_l;$$

moreover, no other two nodes are connected by such a path. Secondly, the kth bit of the two nodes differ if there are nodes $\nu_1, \ldots, \nu_n$ such that there is a path

$$\nu_c \xrightarrow{rootstart_k} \nu_1 \xrightarrow{wired} \ldots \xrightarrow{wired} \nu_n \xrightarrow{end_k} \nu_l;$$

moreover, no other two nodes are connected by such a path (to see this, note that *wired* goes both ways inside the tree whence it is possible to make a back-and-forth step where necessary).

This allows us to assign *head* to all leaf nodes where a path of the first kind exists **for every** k as expressed by the following rule:

$$\bigwedge_{k=1}^{n}\left(rootstart_k(x, x_{k,1}) \wedge \left(\bigwedge_{i=1}^{n-2} wired(x_{k,i}, x_{k,i+1})\right) \wedge end_k(x_{k,n-1}, y)\right) \rightarrow head(y). \quad (23)$$

Likewise, we can assign *nohead* to all leaf nodes where a path of the second kind exists **for some** k, thus we add for every k ranging from 1 to n a separate rule of the following kind:

$$rootstart_k(x, x_1) \wedge \left(\bigwedge_{i=1}^{n-1} wired(x_i, x_{i+1})\right) \wedge end_k(x_n, y) \rightarrow nohead(y). \quad (24)$$

Now that we have an indicator of the head position in the memory tree, we can enforce that every configuration element is indeed assigned the $symbol_\gamma$ predicate whenever the symbol γ is found at the current head position:

$$\left(\bigwedge_{i=0}^{n-1} child(x_i, x_{i+1})\right) \wedge symbol_\gamma(x_n) \wedge head(x_n) \rightarrow symbol_\gamma(x_0). \tag{25}$$

The last bit of the alternating Turing machine functionality that needs to be taken care of is memory evolution: a symbol stored in memory changes according to the transition relation if and only if the head is at the corresponding position. In our encoding this means that for all *nohead*-assigned leaf nodes of a configuration's memory tree, their stored symbol has to be propagated to the corresponding leaf nodes of all direct successors' memory trees. Again we exploit structural properties to connect the corresponding leaf nodes of two subsequent configurations' memory trees:

Let ν_c and ν'_c be two configuration elements such that ν'_c represents a direct successor of ν_c. Let ν_l be a leaf node of ν_c's memory tree and let ν'_l be a leaf node of ν'_c's memory tree. Let the kth bit of ν_l's address and the kth bit of ν'_l's address coincide. Then – and only then – there are nodes $\nu_1, \ldots, \nu_{2n+6}$ such that there is a path

$$\nu_l \xrightarrow{start_k} \nu_1 \xrightarrow{wired} \ldots \xrightarrow{wired} \nu_{2n+6} \xrightarrow{end_k} \nu'_l$$

where fw holds for ν_{n+3}.

This justifies to transfer the stored symbol from any non-head-leaf to all leaf nodes to which it is simultaneously connected by such paths **for every** k:

$$\begin{aligned}\bigwedge_{k=1}^{n}\Bigg(start_k(x, x_{k,1}) \wedge \left(\bigwedge_{i=1}^{2n+5} wired(x_{k,i}, x_{k,i+1})\right) \wedge end_k(x_{k,2n+6}, y)\Bigg) \wedge \\ \left(\bigwedge_{k=1}^{n} fwd(x_{k,n+3})\right) \wedge symbol_\gamma(x) \wedge nohead(x) \quad \rightarrow \quad symbol_\gamma(y).\end{aligned} \tag{26}$$

Of course, we also need to take care to assign the proper symbol (which is determined by the transition by which the current configuration has been reached) to the leaf node of the previous configuration's head position. To this end, we add for every $\delta = (q, \gamma, q', \gamma', D) \in \Delta$ the rule

$$\begin{aligned}head(x) \wedge \bigwedge_{k=1}^{n}\Bigg(start_k(x, x_{k,1}) \wedge \left(\bigwedge_{i=1}^{2n+5} wired(x_{k,i}, x_{k,i+1})\right) \wedge end_k(x_{k,2n+6}, z_n)\Bigg) \wedge \\ \left(\bigwedge_{k=1}^{n} fwd(x_{k,n+3})\right) \wedge trans_\delta(z, z_0) \wedge \left(\bigwedge_{i=0}^{n-1} child(z_i, z_{i+1})\right) \rightarrow symbol_\gamma(z_n).\end{aligned} \tag{27}$$

Finally, we have to ensure that the initial configuration and its memory tree carry all the necessary information. We have to initialize the head position address to 0 by adding to F the facts

$$\{lbit_1(init), \ldots, lbit_n(init)\}. \tag{28}$$

Moreover, all tape cells initially contain the blank symbol $\square$, which we achieve by extending $\mathcal{R}$ by the rule

$$\mathit{firstconf}(x_0) \wedge \left(\bigwedge_{k=0}^{n-1} \mathit{child}(x_k, x_{k+1}) \right) \rightarrow \mathit{symbol}_{\square}(x_n). \tag{29}$$

Concluding, we have just built F, $\mathcal{R}$ and Q with the desired properties. Moreover, $\mathcal{R}$ consists of only *fr1* rules and does not contain any constant. This concludes our argument that the combined complexity of CQ entailment over *fr1* rules is 2ExpTime-hard, even in the case where no constants show up in the rules.

Theorem 34. *Conjunctive query entailment for constant-free fr1 rules with bounded predicate arity is* 2ExpTime*-hard.*

5.5 Combined Complexity of Weakly Guarded Frontier-One Rules is 2ExpTime-hard

Our last hardness result will be established along the same lines as the preceding one, namely by a reduction from the word problem of an alternating Turing machine with exponential space. In fact, we will also reuse part of the reduction and arguments presented in the previous section. In particular, we assume everything up to formula (14) as before except for the following modifications:

- Remove from Rule (2) all atoms built from the predicates *wire* and *fw*.
- Replace the Rule (5) with the following rules:
  - for every $\delta = (q, \gamma, q', \gamma', D) \in \Delta$ the rule

    $$\mathit{trans}_\delta(x, y) \wedge \mathit{accept}(y) \rightarrow \mathit{accept}_\delta(x)$$

  - the rule

    $$\bigwedge_{\delta=(q,\gamma,q',\gamma',D)\in\Delta} \mathit{accept}_\delta(x) \rightarrow \mathit{accept}(x)$$

    Thereby, we introduce a fresh predicate accept_δ for every $\delta = (q, \gamma, q', \gamma', D) \in \Delta$. Clearly this set of rules has the same consequences as the previous Rule (5), however it consists merely of *gfr1* rules.

This puts the ATM's "state space" and transition relations into place. Now we turn to the task of encoding the exponential tape. As opposed to the previous encoding, we now exploit that we can use predicates of arbitrary arity. Thus, for every $\gamma \in \Gamma$ we introduce an n+1-ary predicate ontape_γ where the first n positions are used for the binary encoding of a tape address and the n+1st position contains the configuration element that this tape

information refers to. Our encoding will ensure that the first n positions of these predicates are non-affected. Likewise we will use n+1-ary predicates *head* and *nohead* to store for each tape position of a configuration if the ATM's head is currently in that position or not. For this purpose, we introduce auxiliary constants to encode whether address bits are high, low, or unknown. Thus we add to F the following atoms:

$$high(1), \quad bit(1), \quad low(0), \quad bit(0).$$

Now, for every k we introduce a binary predicate bit_k (whose second position is non-affected) with the intention to let $bit_k(x, 0)$ hold whenever $lbit_k(x)$ holds and to also have $hbit_k(x)$ imply $bit_k(x, 1)$, which is achieved by the following two rules:

$$lbit_k(x) \wedge low(z) \rightarrow bit_k(x, z),$$

$$hbit_k(x) \wedge high(z) \rightarrow bit_k(x, z).$$

Moreover, we make sure that the binary encoding of the head position's address that we find attached to the configuration elements through the bit_k predicates is transferred into the *head* predicate as stated above:

$$conf(x) \wedge \bigwedge_{k=1}^{n} bit_k(x, z_k) \rightarrow head(z_1, \ldots, z_n, x).$$

Additionally, we make sure that for all **other** binary addresses $z_1 \ldots z_n$ the according *nohead* atoms hold, by adding for every i in the range from 1 to n the rule

$$conf(x) \wedge bit_i(x, w) \wedge other(w, z_i) \bigwedge_{k=1}^{n} bit(z_k) \rightarrow nohead(z_1, \ldots, z_n, x).$$

Furthermore, we have to ensure that the symbol γ found at any configuration's head position (expressed by the corresponding $ontape_\gamma$ atom) is also directly attached to that configuration by the according unary $symbol_\gamma$ atom:

$$head(z_1, \ldots, z_n, x) \wedge ontape_\gamma(z_1, \ldots, z_n, x) \rightarrow symbol_\gamma(x).$$

Also, the symbol γ' written to the tape at the previous configuration's head position as a result of some transition δ can be realized easily:

$$head(z_1, \ldots, z_n, x) \wedge trans_\delta(x, y) \rightarrow ontape_{\gamma'}(z_1, \ldots, z_n, x)$$

On the other hand, all previous nohead-positions of the tape will keep their symbol, as made sure by the rules (for all $\gamma \in \Gamma$):

$$nohead(z_1, \ldots, z_n, x) \wedge ontape_\gamma(z_1, \ldots, z_n, x) \wedge trans_\delta(x, y) \rightarrow ontape_\gamma(z_1, \ldots, z_n, y).$$

To ensure that the initial configuration and its tape carry all the necessary information, we initialize the head position address to 0 by adding to F, as in the previous section, the facts:

$$\{lbit_1(init), \ldots, lbit_n(init)\}. \tag{30}$$

To make sure that all tape cells initially contain the blank symbol $\square$, we extend $\mathcal{R}$ by the rule:

$$\mathit{firstconf}(x) \wedge \bigwedge_{k=1}^{n} bit(z_k) \rightarrow ontape_{\square}(z_1, \ldots, z_n, x). \tag{31}$$

Concluding, we have just built F, $\mathcal{R}$ and Q with the desired properties. Moreover, $\mathcal{R}$ consists of only *wgfr1* rules. Thus we arrive at the desired theorem.

Theorem 35. *The combined complexity of CQ entailment over constant-free wgfr1 rules of unbounded arity is* 2ExpTime*-hard.*

6. Body-Acyclic *fg* and *fr1* Rules

In this section, we study the complexity of frontier-guarded rules with an acyclic body. The acyclicity notion considered here is a slight adaptation of *hypergraph acyclicity* stemming from database theory. We will show that body-acyclic *fg* rules coincide with guarded rules: indeed, a body-acyclic *fg* rule can be linearly rewritten as a set of guarded rules, and a guarded rule is a special case of body-acyclic rule.

Let us consider the *hypergraph* naturally associated with a set of atoms S: its set of nodes is in bijection with $terms(S)$ and its multiset of hyperedges is in bijection with S, with each hyperedge being the subset of nodes assigned to the terms of the corresponding atom.

To simplify the next notions, we first proceed with some normalization of a set of *fg* rules, such that all rules have an empty frontier (so-called "disconnected rules" (Baget et al., 2010)) or a "variable-connected" body:

1. Let R be a *fg* rule with a non-empty frontier and let $\mathcal{B}$ be the hypergraph associated with $body(R)$. Split each node in $\mathcal{B}$ assigned to a constant into as many nodes as hyperedges it belongs to (thus each constant node obtained belongs to a single hyperedge); let $\mathcal{B}'$ be the hypergraph obtained; let C_f be the connected component of $\mathcal{B}'$ that contains the frontier guard(s); if there are several frontier guards, they are all in C_f.
2. Let $R_0 = \mathcal{B}' \setminus C_f \rightarrow p_0$, where p_0 is a new nullary predicate.
3. Let $R_f = C_f \cup \{p_0\} \rightarrow head(R)$.

Let $(F, \mathcal{R}, Q)$ be an instance of the entailment problem, where $\mathcal{R}$ is a set of *fg* rules. All non-disconnected rules from $\mathcal{R}$ are processed as described above, which yields an equivalent set of *fg* rules. Let us denote this set by $\mathcal{R}_{disc} \cup \mathcal{R}'$, where R_{disc} is the set of disconnected rules, i.e., initial disconnected rules and obtained rules of form R_0. The rules in $\mathcal{R}_{disc}$ are integrated into F as described in (Baget et al., 2010), which can be performed with

$|\mathcal{R}_{disc}|$ calls to an oracle solving the entailment problem for *fg* rules. Briefly, for each $R_d = (B_d, H_d) \in \mathcal{R}_{disc}$, it is checked whether $F, \mathcal{R}' \models B_d$: if yes, H_d is added to F and R_d is removed from $\mathcal{R}_{disc}$; the process is repeated until stability of $\mathcal{R}_{disc}$. Let F' the fact obtained: for any BCQ Q, $F, \mathcal{R} \models Q$ iff $F', \mathcal{R}' \models Q$. From now on, we thus assume that all *fg* rules have a non-empty frontier and their body is "variable-connected", i.e., the associated hypergraph is connected and cannot be disconnected by the above step 1.

The acyclicity of a hypergraph $\mathcal{H}$ is usually defined with respect to its so-called dual graph, whose nodes are the hyperedges of $\mathcal{H}$ and edges encode non-empty intersections between hyperedges of $\mathcal{H}$. We define below a notion close to the dual graph, that we call *decomposition graph* of a set of atoms. In a decomposition graph, guarded atoms are grouped together with one of their guard into a single node, and constants are not taken into account in atom intersections (it follows that the associated acyclicity notion is slightly more general than hypergraph acyclicity).

Definition 40 (Decomposition Graph)**.** *Let S be a set of atoms. A* decomposition graph *of S is an undirected labeled graph $D_S = (V, E, \mathsf{atoms}, \mathsf{vars})$, where V is the set of nodes, E is the set of edges, atoms and vars are labeling mappings of nodes and of edges respectively, such that:*

- *Let $\{C_1, \dots C_p\}$ be a partition of S such that in each C_i there is an atom that guards the other atoms of C_i, with p being minimal for this property. Then $V = \{v_1, \dots v_p\}$ and for $1 \leq i \leq p$, $\mathsf{atoms}(v_i) = C_i$.*
- *For $1 \leq i, j \leq p$, $i \neq j$, there is an edge $v_i v_j$ if C_i and C_j share a variable; $\mathsf{vars}(v_i v_j) = \mathsf{vars}(C_i) \cap \mathsf{vars}(C_j)$.*

Several decomposition graphs can be assigned to S, however they have all the same structure and the same labeling on edges. The only difference between them comes from the choice of a guard when an atom is guarded by several guards with incomparable sets of variables. Now, considering the decomposition graph instead of the dual graph, the acyclicity of a set of atoms is then defined similarly to that of a hypergraph.

Definition 41 (Acyclicity of an Atom Set, Body-Acyclic *fg* Rule)**.** *Let S be a set of atoms and D_S be a decomposition graph of S. An edge $v_i v_j$ in D_S is said to be* removable *if there is another path λ between v_i and v_j in D_S such that for each each edge $v_k v_l$ in λ, $\mathsf{vars}(v_i v_j) \subseteq \mathsf{vars}(v_k v_l)$. An* acyclic covering *of S is a forest obtained from D_S by removing removable edges only. S is said to be* acyclic *if has an acyclic covering. An *fg* rule R is said to be* body-acyclic (ba) *if $\mathsf{body}(R)$ is acyclic.*

Example 20. *Let $S = \{p_1(x), p_2(x, u), p_2(y, z), p_3(y, z, u), p_2(u, v), p_3(u, v, x)\}$. D_S has set of nodes $\{v_1, v_2, v_3\}$, with $\mathsf{atoms}(v_1) = \{p_1(x), p_2(x, u)\}$, $\mathsf{atoms}(v_2) = \{p_2(y, z), p_3(y, z, u)\}$ and $\mathsf{atoms}(v_3) = \{p_2(u, v), p_3(u, v, x)\}$, and all edges between these nodes, with $\mathsf{vars}(v_1 v_2) = \mathsf{vars}(v_2 v_3) = \{u\}$ and $\mathsf{vars}(v_1 v_3) = \{x, u\}$. An acyclic covering of S is obtained by removing edge $v_1 v_2$ or $v_2 v_3$.*

Let us point out that a set of atoms is acyclic according to the above definition if and only if the associated existentially closed conjunctive formula belongs to the guarded fragment of first-order logic (see *acyclic guarded covering* in (Kerdiles, 2001) and (Chein &

Mugnier, 2009) for details about this equivalence). Note also that the decomposition graph associated with the body of a guarded rule is restricted to a single node. Thus, guarded rules are trivially ba-*fg* rules.

Given a set of atoms S, checking whether it is acyclic, and if so, outputing one of its acyclic coverings can be performed in linear time (from (Maier, 1983) about the computation of a join tree in databases).

Let R be a variable-connected ba-*fg* rule and let T be an acyclic covering of *body(R)*, which is thus a tree. Let $\{v_1, ..., v_p\}$ be the nodes in T and let v_r be a node such that $atoms(v_r)$ contains a frontier guard. T is considered as rooted in v_r, which yields a direction of its edges from children to parents: a directed edge (v_i, v_j) is from a child to its parent. R is translated into a set of guarded rules $\{R_1, \dots R_p\}$ as follows:

- To each edge (v_i, v_j) is assigned the atom $a_i = q_i(vars(v_i v_j))$, where q_i is a new predicate;
- To each node v_i, $i \neq r$, is assigned the rule:
  $R_i = atoms(v_i) \cup \{a_k | v_k \text{ child of } v_i\} \rightarrow a_i$
- To the node v_r is assigned the rule:
  $R_r = atoms(v_r) \cup \{a_k | v_k \text{ child of } v_r\} \rightarrow head(R)$

Note that this translation is the identity on guarded rules.

Example 20 (Contd.) Let $R = p_1(x) \wedge p_2(x,u) \wedge p_2(y,z) \wedge p_3(y,z,u) \wedge p_2(u,v) \wedge p_3(u,v,x) \rightarrow head(R)$, with $fr(R) = \{u, v\}$. Consider the acyclic covering with set of nodes $\{v_1, v_2, v_3\}$, with $atoms(v_1) = \{p_1(x), p_2(x,u)\}$, $atoms(v_2) = \{p_2(y,z), p_3(y,z,u)\}$ and $atoms(v_3) = \{p_2(u,v), p_3(u,v,x)\}$, and set of edges $\{v_1v_3, v_2v_3\}$. On has $vars(v_1v_3) = \{u\}$ and $vars(v_1v_3) = \{x, u\}$. v_3 is the root. The obtained guarded rules are:
$R_1 = p_1(x) \wedge p_2(x,u) \rightarrow q_1(x,u)$
$R_2 = p_2(y,z) \wedge p_3(y,z,u) \rightarrow q_2(u)$
$R_3 = p_2(u,v) \wedge p_3(u,v,t) \wedge q_1(x,u) \wedge q_2(u) \rightarrow head(R)$

Property 36. *Guarded rules and ba-fg rules are semantically equivalent in the following sense: a guarded rule is a ba-fg rule and any set of ba-fg rules can be translated into a semantically equivalent set of guarded rules.*

The above translation is polynomial in the size of the rule and arity-preserving. Thus, complexity results on guarded rules apply to ba-*fg* rules. In particular ba-*fg* rules are ExpTime-complete for bounded-arity combined complexity, while *fg* rules are 2ExpTime-complete.

Actually the ExpTime lower bound already holds for combined complexity with arity bounded by 2. Indeed, standard reasoning in the much weaker description logic reduced normalized Horn-$\mathcal{ALC}$ (cf. Section 5.2), which is a fragment of ba-*gfr1* rules with maximal arity of 2, is already ExpTime-Hard, as cited in Theorem 29. It follows that ba-*gfr1* rules are ExpTime-Hard for bounded-arity in combined complexity. One could have expected ba-frontier-1 rules to be simpler than ba-*fg* rules. In fact, they have the same data complexity

and the same bounded-arity combined complexity. The only remaining question, for which we have no answer yet, is whether they are simpler in the unbounded arity case.

Finally, let us point out that the acyclicity of rule bodies alone is not enough to guarantee a lower complexity, and even decidability: that the head of a rule shares variables with only one node of the decomposition graph (thus, that the frontier is guarded) is crucial. Without this assumption, the entailment problem remains undecidable. [12]

7. Related Work

In this section, discuss the relationship of the existential rule fragments considered here with another major paradigm in logic-based knowledge representation: description logics. Also, we will point out similarities of the techniques applied in the presented algorithm with reasoning approaches established for other logics.

7.1 Relationships to Horn Description Logics and their Extensions

The relationship of description logics and existential rules has often been recognized. In particular Horn-DLs (Hustadt, Motik, & Sattler, 2005; Krötzsch et al., 2007, 2013) share many properties with existential rules such as the existence of a (homomorphically unique) canonical model. Crucial differences between the two approaches are that (1) as opposed to DLs, existential rules allow for predicates of arity greater than two as well as for the description of non-tree shaped terminological information and (2) as opposed to existential rules, expressive DLs allow for a tighter integration of cardinality constraints to a degree (at least currently) unachieved by existential rules.

In the following, we will point out which Horn-DLs are subsumed by which existential rules fragments. We will refrain from providing full translations and restrict ourselves to examples that provide the underlying intuition.

The description logic $\mathcal{EL}$ essentially allows for encoding implications of tree-shaped substructures in a model. For instance the statement "Everybody who has a caring mother and a caring father has a nice home" can be expressed by the $\mathcal{EL}$ axiom $\exists \mathit{hasMother}.\mathit{Caring} \sqcap \exists \mathit{hasFather}.\mathit{Caring} \sqsubseteq \exists \mathit{hasHome}.\mathit{Nice}$ which is equivalent to the existential rule

$$\mathit{hasMother}(x, y) \wedge \mathit{Caring}(y) \wedge \mathit{hasFather}(x, z) \wedge \mathit{Caring}(z) \rightarrow \mathit{hasHome}(x, w) \wedge \mathit{Nice}(w).$$

Horn-$\mathcal{ALC}$ is more expressive than $\mathcal{EL}$ in that it allows to express some sort of universal quantification such as in "Whenever some caring person has children, all of them are happy" denoted by $\mathit{Caring} \sqsubseteq \forall \mathit{hasChild}$ which corresponds to the existential rule $\mathit{Caring}(x) \wedge \mathit{hasChild}(x, y) \rightarrow \mathit{Happy}(y)$.

It is not hard to see that the Horn-DLs $\mathcal{EL}$ and Horn-$\mathcal{ALC}$ are captured by *fr1* rules; they can even be linearly rewritten into *gfr1* rules when auxiliary predicates are allowed (as it is often done when normalizing DL knowledge bases). This still holds when these DL languages are extended by inverses (indicated by adding an $\mathcal{I}$ to the name of the DL: $\mathcal{ELI}$,

12. See for instance (Baget & Mugnier, 2002), which provides a reduction from the word problem in a semi-Thue system, known to be undecidable, to the CQ entailment problem with existential rules (in a conceptual graph setting): this reduction yields existential rules with predicate arity bounded by 2, a body restricted to a path and a frontier of size 2.

Horn-$\mathcal{ALCI}$) and/or nominals (indicated by adding an $\mathcal{O}$). In the latter case, constants must occur in the rules). For instance, the $\mathcal{ELIO}$ proposition "Everybody who is born in Germany likes some soccer team which has a member who is also a member of the the German national team", written in DL notation

$$\exists bornin.\{germany\} \sqsubseteq \exists likes.(SoccerTeam \sqcap \exists hasMember.\exists hasMember^{-}.\{gnt\})$$

, in existential rules form

$$bornin(x, germany) \rightarrow likes(x, y) \wedge SoccerTeam(y) \wedge hasMember(y, z) \wedge hasMember(gnt, z)$$

Role hierarchies ($\mathcal{H}$), allow to express generalization/specialization relationships on binary predicates (such as *fatherOf* implying *parentOf*). Adding this feature to any of the abovementioned description logics requires to use rules with frontier size 2 and thus leads outside the frontier-one fragment. Still a linear translation into g rules remains possible.[13]. Going further to DLs that feature functionality or transitivity of binary predicates leads to existential rule fragments which are not longer guarded in any way considered here. The definition of existential rule fragments capturing these expressive description logics is subject of ongoing research.

Diverse proposals have been made to overcome the structural restrictions of DLs, i.e. to allow to express non-tree-shaped relationships in the terminological knowledge. *Description graphs* (Motik, Grau, Horrocks, & Sattler, 2009b) constitute one of these endeavors, where the existentially quantified structure in the head of a DL axiom is allowed to be arbitrarily graph-shaped and, additionally, there are datalog rules operating locally on these graph structures. It is straight-forward that the extension of any DL up to Horn-$\mathcal{ALCHIO}$ by description graphs can be coded into g existential rules.

Another suggestion made to allow for non-tree shaped structures in both body and head of a DL axiom is to introduce *DL-safe variables*, that is, variables that are only allowed to be bound to "named individuals" (i.e., domain elements denoted by constants). In a setting where each statement can carry either exclusively safe variables or exclusively non-safe ones, this can be captured by the notion of *DL-safe rules* (Motik, Sattler, & Studer, 2005). A more liberal approach is that of the so-called *nominal schemas* (Krötzsch, Maier, Krisnadhi, & Hitzler, 2011; Krötzsch & Rudolph, 2014), where the two types of variables can occur jointly in the same statement. In both cases, the wg fragment captures these extensions when applied to DLs up to Horn-$\mathcal{ALCHIO}$. Note that there is a direct correspondence between non-affected positions in existential rules and DL-safe variables: both can only carry domain elements corresponding to elements present in the initial data.

7.2 Pattern- or Type-Based Reasoning

For many logics, reasoning algorithms as well as related complexity arguments are based on the notion of types. On an intuitive level, types represent "configurations" which may occur in a model. In the easiest case (as in some description logics), such configurations might be sets of unary predicates $\{p_1, \ldots, p_n\}$, where $\{p_1, \ldots, p_n\}$ occurring in a model just

13. It might, however, be noteworthy that it is possible to come up with a polynomial translation into *gfr1* rules by materializing the subsumption hierarchy of the binary predicates upfront and, whenever a binary atom is created by a rule "creating" all the subsumed atoms at the same time.

means that there is an individual a that is in the extension of every p_i. More complex notions of types may refer to more than just one individual, leading to notions like 2-types, also known as dominoes (Rudolph, Krötzsch, & Hitzler, 2012), star-types (Pratt-Hartmann, 2005), mosaics (Blackburn, van Benthem, & Wolter, 2006), or types based on trees (Rudolph & Glimm, 2010). Often, reasoning in a logic can be carried out by only considering the (multi-)set of types that are realized in a model. Typical reasoning strategies may then compute the set of these types bottom-up (as in tableaux with anywhere-blocking), top-down (as in type-elimination-based procedures), or describe their multiplicity by means of equational systems. The applicability of such strategies guarantees decidability whenever the overall set of possible types is finite. It is not hard to see that, in our case, abstract patterns can be seen as "graph types" where the bound on the tree-width and the finiteness of the vocabulary guarantee the finiteness of the set of types, and therefore the effectiveness of the applied blocking strategy.

7.3 Consequence-Driven Reasoning

Our idea of the saturation of pattern rules in Section ... has many similarities with the approach of consequence-driven reasoning in description logics. In both cases, logical sentences that are consequences of a given theory are materialized. To see this, one should be aware that every evolution rule $\mathbb{P}_1 \rightsquigarrow \mathbb{P}_2$ corresponds to an existential rule

$$\bigwedge_{(G,\pi)\in\mathbb{P}_1} \pi^{\text{safe}}(G) \rightarrow \bigwedge_{(G,\pi)\in\mathbb{P}_2} \pi^{\text{safe}}(G)$$

and every creation rule $\mathbb{P}_1 \rightsquigarrow \lambda.\mathbb{P}_2$ corresponds to an existential rule

$$\bigwedge_{(G,\pi)\in\mathbb{P}_1} \pi^{\text{safe}}(G) \rightarrow \bigwedge_{(G,\pi)\in\mathbb{P}_2} \pi^{\text{safe}}(\lambda(G)).$$

It can then be readily checked that the deduction calculus presented in Fig. 5 is indeed sound. As such, the presented algorithm has indeed similarities with type-based consequence driven reasoning approaches as, e.g., layed out by (Kazakov, 2009) and (Ortiz, Rudolph, & Simkus, 2010, 2011). The crucial difference here is that the mentioned works use only 1-types, whereas the patterns defined characterize larger "clusters" of elements.

7.4 Tableaux and Blocking

It is well-known that the chase known from databases has many commonalities with the semantic tableau method in FOL (Beth, 1955; Smullyan, 1968), which has also been used in many other logics, most notably DLs (Baader & Sattler, 2001; Horrocks & Sattler, 2007). Note that the generic semantic tableaux for first order logic only gives rise to a semidecision procedure. In order to obtain a decision procedure for a restricted logic, termination needs to be guaranteed, typically through establishing a tree(-like) model property and the detection of repetitions in the course of the tableaux construction, leading to the idea of *blocking* as soon as repeating types occur (depending on the expressivity of the logic, 1-types, 2-types or even larger types have to be considered). Clearly, the blocking technique used by us in the construction of the full blocked tree can be seen as a pattern-based anywhere blocking.

7.5 Relationships to other work on guarded existential rules

As already mentioned, guarded and weakly-guarded rules were introduced in (Calì et al., 2008, 2009; Calì, Gottlob, & Kifer, 2013). A fundamental notion used to bound the depth of the breadth-first saturation (with a bound depending on $\mathcal{R}$ and Q) is that of the type of a guard atom in the saturation (a guard atom in $\alpha_\infty(F, \mathcal{R})$ is the image of a rule guard by a rule application, and the type of an atom a is the set all atoms in $\alpha_\infty(F, \mathcal{R})$ with arguments included in $terms(a)$). This notion has some similarities with our bag patterns, without being exactly the restriction of bag patterns to the guarded case.

The notion of affected position/variable was refined into that of *jointly affected* position/variable in (Krötzsch & Rudolph, 2011). This yields the new classes of *jointly guarded* rules and *jointly frontier-guarded* rules, which respectively generalize *wg* rules and *wfg* rules. Since these new classes are *gbts*, our results apply to them. In particular, the data and combined complexities of jointly frontier-guarded rules directly follow from those of *gbts* and *wfg*: ExpTime-complete data complexity, and 2ExpTime-complete combined complexity (in both bounded and unbounded predicate arity cases). Since *wg* rules have the same complexities as *wfg* rules for data complexity and combined complexity with bounded arity, these complexities also apply to jointly guarded rules; for combined complexity with unbounded arity, the bounds are not tight (ExpTime-hardness from the result on *wg* and 2ExpTime-membership from the result on *wfg*). Note that (Krötzsch & Rudolph, 2011) also provides a further generalization, namely *glut frontier-guarded*, which is *bts*, but not *gbts* nor *fes*.

Combinations of the *gbts* family with other families of rules have been proposed, by restricting possible interactions between rules of the different kinds. In (Baget et al., 2011), conditions expressed on the strongly connected components of a graph of rule dependencies allow to combine *gbts*, *fes* and *fus* sets of rules. In (Gottlob, Manna, & Pieris, 2013), a notion called tameness allows to combine guarded rules with sticky rules (an expressive *fes* concrete class of rules) by restricting the interactions between the sticky rules and the guard atoms in the guarded rules.

Finally, let us cite some very recent work related to the guarded family of existential rules. The expressiveness of the *wg* and *wfg* fragments was studied in (Gottlob, Rudolph, & Simkus, 2014). Other work has analyzed the complexity of entailment with guarded rules extended with disjunction (Bourhis, Morak, & Pieris, 2013) or with stable negation (Gottlob, Hernich, Kupke, & Lukasiewicz, 2014b).

7.6 Combined Approach

The combined approach (Lutz et al., 2009; Kontchakov, Lutz, Toman, Wolter, & Zakharyaschev, 2010), designed for $\mathcal{EL}$ and the DL-Lite familly, share some similarities with our approach. The combined approach is a two-step process. First, some materialization is performed. In order to ensure finiteness of this step, an over-specialization of the canonical model is thus computed. This over-specialization requires thus a rewriting of the query, in order to recover soundness. This rewriting may require the ontology. By comparison, our approach computes a materialization that is less specific than the saturation. It is thus the completeness that has to be recovered, through a change of the querying operation, which is not a simple homomorphism anymore, but is based on the notion of APT-homomorphism.

8. Conclusion

We have introduced the notion of greedy bounded-treewidth sets of existential rules that subsumes guarded rules, as well as their known generalizations, and gives rise to a generic and worse-case optimal algorithm for deciding conjunctive query entailment. Moreover, we have classified known *gbts* subclasses with respect to their combined and data complexities.

The existential rule framework is young and combines techniques from different research fields (such as databases, description logics, rule-based languages). A lot of work is still to be done to deepen its understanding, design efficient algorithms and develop its applications.

It remains an open question whether *gbts* is recognizable. We conjecture that the answer is yes. However, even if this question is interesting from a theoretical viewpoint, we recall that *gbts* is not more expressive than *wfg*, and furthermore, any *gbts* set of rules can be *polynomially* translated into a *wfg* set of rules, while preserving entailment (Section 3). Moreover, we built a reduction from the entailment problem, where $\mathcal{R}$ is *fg* and constant-free, to the problem of checking if some rule set $\mathcal{R}'$ is *gbts* (this reduction is not included in this paper, since it is only one step in the study of the recognizability issue). Hence, we know that determining if some rule set is *gbts* is significantly harder than for *wfg*, where this check can be done in polynomial time.

Future work will aim at adding rules expressing restricted forms of equality and of useful properties such as transitivity into this framework, while preserving decidability, and the desirable PTime data complexity of *fg* rules.

We have shown in Section 4.6 that the PatSat algorithm can be adapted to run with optimal worst-case complexity for fragments of the *gbts* family. An important research line is the optimization, implementation and practical evaluation of this algorithm and its variants. To the best of our knowledge, currently existing prototypes processing existential rules follow a backward chaining approach, which involves rewriting the query into a union of CQs; hence, termination of the query rewriting process is ensured on *fus* rules (Gottlob, Orsi, & Pieris, 2014a; König, Leclère, Mugnier, & Thomazo, 2014)

Acknowledgments

Michaël Thomazo aknowledges support from the Alexander von Humboldt foundation.

Bibliography

Abiteboul, S., Hull, R., & Vianu, V. (1994). *Foundations of Databases*. Addison Wesley.

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

Andréka, H., Németi, I., & van Benthem, J. (1998). Modal Languages and Bounded Fragments of Predicate Logic. *J. of Philosophical Logic*, *27*, 217–274.

Andréka, H., van Benthem, J., & Németi, I. (1996). Modal languages and bounded fragments of FOL. Research report ML-96-03, Univ. of Amsterdam.

Baader, F. (2003). Least common subsumers and most specific concepts in a description logic with existential restrictions and terminological cycles. In Gottlob, G., & Walsh,

T. (Eds.), *Proceedings of the 18th International Joint Conference on Artificial Intelligence (IJCAI'03)*, pp. 325–330. Morgan Kaufmann.

Baader, F., Calvanese, D., McGuinness, D., Nardi, D., & Patel-Schneider, P. (Eds.). (2007). *The Description Logic Handbook: Theory, Implementation, and Applications* (Second edition). Cambridge University Press.

Baader, F., & Sattler, U. (2001). An overview of tableau algorithms for description logics. *Studia Logica*, *69*(1), 5–40.

Baget, J.-F., Leclère, M., & Mugnier, M.-L. (2010). Walking the Decidability Line for Rules with Existential Variables. In *Principles of Knowledge Representation and Reasoning: Proceedings of the Twelfth International Conference, KR 2010, Toronto, Ontario, Canada, May 9-13, 2010*. AAAI Press.

Baget, J.-F., Leclère, M., Mugnier, M.-L., & Salvat, E. (2009). Extending Decidable Cases for Rules with Existential Variables. In *IJCAI 2009, Proceedings of the 21st International Joint Conference on Artificial Intelligence, Pasadena, California, USA, July 11-17, 2009*, pp. 677–682.

Baget, J.-F., Leclère, M., Mugnier, M.-L., & Salvat, E. (2011). On rules with existential variables: Walking the decidability line. *Artificial Intelligence*, *175*(9-10), 1620–1654.

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

Baget, J.-F., Mugnier, M.-L., Rudolph, S., & Thomazo, M. (2011). Walking the Complexity Lines for Generalized Guarded Existential Rules. In *IJCAI 2011, Proceedings of the 22nd International Joint Conference on Artificial Intelligence, Barcelona, Catalonia, Spain, July 16-22, 2011*, pp. 712–717.

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

Beeri, C., & Vardi, M. (1984). A Proof Procedure for Data Dependencies. *Journal of the ACM*, *31*(4), 718–741.

Beth, E. W. (1955). Semantic entailment and formal derivability. *Mededelingen van de Koninklijke Nederlandse Akademie van Wetenschappen, Afdeling Letterkunde*, *18*(13), 309–342.

Blackburn, P., van Benthem, J. F. A. K., & Wolter, F. (Eds.). (2006). *Handbook of Modal Logic*, Vol. 3 of *Studies in Logic and Practical Reasoning*. Elsevier Science.

Bourhis, P., Morak, M., & Pieris, A. (2013). The impact of disjunction on query answering under guarded-based existential rules. In *IJCAI 2013, Proceedings of the 23rd International Joint Conference on Artificial Intelligence, Beijing, China, August 3-9, 2013*.

Calì, A., Gottlob, G., & Kifer, M. (2008). Taming the Infinite Chase: Query Answering under Expressive Relational Constraints. In *Principles of Knowledge Representation and Reasoning: Proceedings of the Eleventh International Conference, KR 2008, Sydney, Australia, September 16-19, 2008*, pp. 70–80.

Calì, A., Gottlob, G., & Kifer, M. (2013). Taming the infinite chase: Query answering under expressive relational constraints. *J. Artif. Intell. Res. (JAIR)*, *48*, 115–174.

Calì, A., Gottlob, G., Lukasiewicz, T., Marnette, B., & Pieris, A. (2010). Datalog+/-: A family of logical knowledge representation and query languages for new applications. In *Proceedings of the 25th Annual IEEE Symposium on Logic in Computer Science, LICS 2010, 11-14 July 2010, Edinburgh, United Kingdom*, pp. 228–242.

Calì, A., Gottlob, G., & Kifer, M. (2013). Taming the infinite chase: Query answering under expressive relational constraints. *J. Artif. Intell. Res. (JAIR)*, *48*, 115–174.

Calì, A., Gottlob, G., & Lukasiewicz, T. (2009). A General Datalog-Based Framework for Tractable Query Answering over Ontologies. In *Proceedings of the Twenty-Eigth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, PODS 2009, June 19 - July 1, 2009, Providence, Rhode Island, USA*, pp. 77–86. ACM.

Calì, A., Lembo, D., & Rosati, R. (2003). On the decidability and complexity of query answering over inconsistent and incomplete databases. In *Proceedings of the Twenty-Second ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, June 9-12, 2003, San Diego, CA, USA*, pp. 260–271.

Calvanese, D., De Giacomo, G., & Lenzerini, M. (1998). On the decidability of query containment under constraints. In *Proceedings of the 17th ACM SIGACT SIGMOD Symposium on Principles of Database Systems (PODS 1998)*, pp. 149–158. ACM Press and Addison Wesley.

Calvanese, D., De Giacomo, G., Lembo, D., Lenzerini, M., & Rosati, R. (2007). Tractable Reasoning and Efficient Query Answering in Description Logics: The DL-Lite Family. *Journal of Automated Reasoning*, *39*(3), 385–429.

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

Chandra, A. K., Kozen, D. C., & Stockmeyer, L. J. (1981b). Alternation. *Journal of the ACM*, *28*(1), 114–133.

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

Courcelle, B. (1990). The Monadic Second-Order Logic of Graphs: I. Recognizable Sets of Finite Graphs. *Inf. Comput.*, *85*(1), 12–75.

Deutsch, A., Nash, A., & Remmel, J. (2008). The chase revisited. In *Proceedings of the Twenty-Seventh ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, PODS 2008, June 9-11, 2008, Vancouver, BC, Canada*, pp. 149–158.

Fagin, R., Kolaitis, P. G., Miller, R. J., & Popa, L. (2005). Data Exchange: Semantics and Query Answering. *Theor. Comput. Sci.*, *336*(1), 89–124.

Gottlob, G., Orsi, G., & Pieris, A. (2014a). Query rewriting and optimization for ontological databases. *ACM Trans. Database Syst.*, *39*(3), 25.

Gottlob, G., Hernich, A., Kupke, C., & Lukasiewicz, T. (2014b). Stable model semantics for guarded existential rules and description logics. In *Principles of Knowledge Representation and Reasoning: Proceedings of the Fourteenth International Conference, KR 2014, Vienna, Austria, July 20-24, 2014*.

Gottlob, G., Manna, M., & Pieris, A. (2013). Combining decidability paradigms for existential rules. *Theory and Practice of Logic Programming*, *13*(4-5), 877–892.

Gottlob, G., Rudolph, S., & Simkus, M. (2014). Expressiveness of guarded existential rule languages. In *Proceedings of the 33rd ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, PODS'14, Snowbird, UT, USA, June 22-27, 2014*, pp. 27–38.

Horrocks, I., & Sattler, U. (2007). A tableau decision procedure for $\mathcal{SHOIQ}$. *J. Autom. Reasoning*, *39*(3), 249–276.

Hustadt, U., Motik, B., & Sattler, U. (2005). Data complexity of reasoning in very expressive description logics. In Kaelbling, L., & Saffiotti, A. (Eds.), *Proc. 19th Int. Joint Conf. on Artificial Intelligence (IJCAI'05)*, pp. 466–471. Professional Book Center.

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

Kazakov, Y. (2009). Consequence-driven reasoning for horn-$\mathcal{SHIQ}$ ontologies. In Boutilier, C. (Ed.), *IJCAI 2009, Proceedings of the 21st International Joint Conference on Artificial Intelligence, Pasadena, California, USA, July 11-17, 2009*, pp. 2040–2045.

Kerdiles, G. (2001). *Saying it with Pictures: a logical landscape of conceptual graphs*. Ph.D. thesis, Univ. Montpellier II / Amsterdam.

König, M., Leclère, M., Mugnier, M., & Thomazo, M. (2014). Sound, complete and minimal ucq-rewriting for existential rules..

Kontchakov, R., Lutz, C., Toman, D., Wolter, F., & Zakharyaschev, M. (2010). The combined approach to query answering in DL-lite. In *Principles of Knowledge Representation and Reasoning: Proceedings of the Twelfth International Conference, KR 2010, Toronto, Ontario, Canada, May 9-13, 2010*.

Krötzsch, M., & Rudolph, S. (2011). Extending Decidable Existential Rules by Joining Acyclicity and Guardedness. In *IJCAI 2011, Proceedings of the 22nd International Joint Conference on Artificial Intelligence, Barcelona, Catalonia, Spain, July 16-22, 2011*, pp. 963–968.

Krötzsch, M., Maier, F., Krisnadhi, A., & Hitzler, P. (2011). A better uncle for owl: nominal schemas for integrating rules and ontologies. In Srinivasan, S., Ramamritham, K., Kumar, A., Ravindra, M. P., Bertino, E., & Kumar, R. (Eds.), *Proceedings of the 20th International Conference on World Wide Web, WWW 2011, Hyderabad, India, March 28 - April 1, 2011*, pp. 645–654. ACM.

Krötzsch, M., & Rudolph, S. (2014). Nominal schemas in description logics: Complexities clarified. In Baral, C., De Giacomo, G., & Eiter, T. (Eds.), *Principles of Knowledge*

Representation and Reasoning: Proceedings of the Fourteenth International Conference, KR 2014, Vienna, Austria, July 20-24, 2014. AAAI Press.

Krötzsch, M., Rudolph, S., & Hitzler, P. (2007). Complexity Boundaries for Horn Description Logics. In *Proceedings of the Twenty-Second AAAI Conference on Artificial Intelligence, July 22-26, 2007, Vancouver, British Columbia, Canada*, pp. 452–457. AAAI Press.

Krötzsch, M., Rudolph, S., & Hitzler, P. (2013). Complexities of Horn description logics. *ACM Trans. Comput. Logic*, *14*(1), 2:1–2:36.

Levy, A. Y., & Rousset, M.-C. (1996). CARIN: A representation language combining Horn rules and description logics. In *Proceedings of the 12th European Conference on Artificial Intelligence (ECAI 1996)*, pp. 323–327.

Lutz, C. (2007). Inverse roles make conjunctive queries hard. In *Proceedings of the 2007 International Workshop on Description Logics (DL2007)*, CEUR-WS.

Lutz, C., Toman, D., & Wolter, F. (2009). Conjunctive Query Answering in the Description Logic $\mathcal{EL}$ Using a Relational Database System. In *IJCAI 2009, Proceedings of the 21st International Joint Conference on Artificial Intelligence, Pasadena, California, USA, July 11-17, 2009*, pp. 2070–2075.

Maier, D. (1983). *The Theory of Relational Databases*. Computer Science Press.

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

Motik, B., Cuenca Grau, B., Horrocks, I., Wu, Z., Fokoue, A., & Lutz, C. (Eds.). (27 October 2009a). *OWL 2 Web Ontology Language: Profiles*. W3C Recommendation. Available at `http://www.w3.org/TR/owl2-profiles/`.

Motik, B., Grau, B. C., Horrocks, I., & Sattler, U. (2009b). Representing ontologies using description logics, description graphs, and rules. *Artif. Intell.*, *173*(14), 1275–1309.

Motik, B., Sattler, U., & Studer, R. (2005). Query answering for OWL DL with rules. *Journal of Web Semantics*, *3*(1), 41–60.

Ortiz, M., Rudolph, S., & Simkus, M. (2010). Worst-case optimal reasoning for the Horn-DL fragments of OWL 1 and 2. In Lin, F., Sattler, U., & Truszczynski, M. (Eds.), *KR*. AAAI Press.

Ortiz, M., Rudolph, S., & Simkus, M. (2011). Query answering in the Horn fragments of the description logics SHOIQ and SROIQ. In Walsh, T. (Ed.), *IJCAI*, pp. 1039–1044. IJCAI/AAAI.

Poggi, A., Lembo, D., Calvanese, D., De Giacomo, G., Lenzerini, M., & Rosati, R. (2008). Linking data to ontologies. *J. Data Semantics*, *10*, 133–173.

Pratt-Hartmann, I. (2005). Complexity of the two-variable fragment with counting quantifiers. *Journal of Logic, Language and Information*, *14*, 369–395.

Rudolph, S. (2011). Foundations of description logics. In Polleres, A., d'Amato, C., Arenas, M., Handschuh, S., Kroner, P., Ossowski, S., & Patel-Schneider, P. F. (Eds.), *Reasoning Web*, Vol. 6848 of *Lecture Notes in Computer Science*, pp. 76–136. Springer.

Rudolph, S. (2014). The two views on ontological query answering. In Gottlob, G., & Pérez, J. (Eds.), *Proceedings of the 8th Alberto Mendelzon Workshop on Foundations of Data Management*, Vol. 1189 of *CEUR Workshop Proceedings*. CEUR-WS.org.

Rudolph, S., & Glimm, B. (2010). Nominals, inverses, counting, and conjunctive queries or: Why infinity is your friend!. *Journal of Artificial Intelligence Research*, *39*, 429–481.

Rudolph, S., Krötzsch, M., & Hitzler, P. (2012). Type-elimination-based reasoning for the description logic $\mathcal{SHIQ}b_s$ using decision diagrams and disjunctive datalog. *Logical Methods in Computer Science*, *8*(1).

Salvat, E., & Mugnier, M.-L. (1996). Sound and Complete Forward and Backward Chainings of Graph Rules. In *Conceptual Structures: Knowledge Representation as Interlingua, 4th International Conference on Conceptual Structures, ICCS '96, Sydney, Australia, August 19-22, 1996, Proceedings*, Vol. 1115 of *LNAI*, pp. 248–262. Springer.

Smullyan, R. M. (1968). *First-order logic*. Dover books on mathematics. Dover.

Thomazo, M. (2013). *Conjunctive Query Answering Under Existential Rules—Decidability, Complexity, and Algorithms*. Ph.D. thesis, Univ. Montpellier 2.

Thomazo, M., Baget, J.-F., Mugnier, M.-L., & Rudolph, S. (2012). A generic querying algorithm for greedy sets of existential rules. In *Principles of Knowledge Representation and Reasoning: Proceedings of the Thirteenth International Conference, KR 2012, Rome, Italy, June 10-14, 2012*.

W3C OWL Working Group (27 October 2009). *OWL 2 Web Ontology Language: Document Overview*. W3C Recommendation. Available at `http://www.w3.org/TR/owl2-overview/`.