UNIVERSITÀ DELLA CALABRIA

UNIVERSITÀ DELLA CALABRIA

Dipartimento di Matematica e Informatica

Dottorato di Ricerca in Matematica e Informatica

XXXIII CICLO

DYADIC TGDs - A NEW PARADIGM FOR ONTOLOGICAL QUERY ANSWERING

Settore Scientifico Disciplinare INF/01

Coordinatore: Ch.mo Prof. Gianluigi Greco

Supervisori: Prof. Marco Manna

Firmato da:Marco Manna
Data: 01/03/2022 10:49:27

Ch.ma Prof.ssa Fran[illegible]uerriero

GIANLUIGI
GRECO
07.03.2022
08:17:17 UTC

GUERRIERO
FRANCESCA
01.03.2022 11:17:03
GMT+01:00

Ch.mo Prof. Nicola Leone

NICOLA
LEONE
01.03.2022
10:53:05
UTC

Dottoranda: Dott.ssa Cinzia Marte

Marte Cinzia
27.02.2022
19:59:50
GMT+00:00

Abstract

Ontology-Based Query Answering (OBQA) consists in querying data–bases by taking ontological knowledge into account. We focus on a logical framework based on existential rules or *tuple generating dependencies* (TGDs), also known as Datalog$^{\pm}$, which collects the basic decidable classes of TGDs, and generalizes several ontology specification languages.

While there exist lots of different classes in the literature, in most cases each of them requires the development of a specific solver and, only rarely, the definition of a new class allows the use of existing systems. This gap between the number of existent paradigms and the number of developed tools, prompted us to define a combination of Shy and Ward (two well-known classes that enjoy good computational properties) with the aim of exploiting the tool developed for Shy.

Nevertheless, studying how to merge these two classes, we have realized that it would be possible to define, in a more general way, the combination of existing classes, in order to make the most of existing systems.

Hence, in this work, starting from the analysis of the two aforementioned existing classes, we define a more general class, named *Dyadic TGDs*, that allows to extend in a uniform and elegant way all the decidable classes, while using the existent related systems. At the same time, we define also a combination of Shy and Ward, named Ward^{+}, and we show that it can be seen as a Dyadic set of TGDs.

Finally, to support the theoretical part of the thesis, we implement a BCQ evaluation algorithm for the class Ward^{+}, that takes advantage of an existing solver developed for Shy.

Sommario

L'Ontology-Based Query Answering (OBQA) consiste nell'interrogare basi di dati tenendo conto delle conoscenze ontologiche. In particolare, ci focalizziamo su un framework logico basato su regole esistenziali o *tuple generating dependencies* (TGD), noto anche come Datalog$^{\pm}$, che include le classi di TGD decidibili.

Nonostante esistano molte classi differenti in letteratura, nella maggior parte dei casi ognuna di esse richiede lo sviluppo di un risolutore specifico e, solo raramente, la definizione di una nuova classe consente l'utilizzo di sistemi esistenti. Questo divario tra il numero di paradigmi esistenti ed il numero di risolutori sviluppati, ci ha spinto a definire una combinazione di Shy e Ward (due classi ben note che godono di buone proprietà) con l'obiettivo di sfruttare il sistema sviluppato per Shy.

Tuttavia, studiando come combinare queste due classi, ci siamo resi conto che sarebbe possibile definire un modo più generale per combinare classi esistenti, con la possibilità di sfruttare al meglio i sistemi già noti.

Pertanto, in questo lavoro, partendo dall'analisi delle due classi sopra citate, abbiamo definito una classe più generale, denominata *Dyadic TGDs*, che permette di estendere in modo uniforme ed elegante tutte le classi decidibili, permettendo l'utilizzo dei sistemi esistenti correlati. Parallelamente, abbiamo definito anche una combinazione di Shy e Ward, chiamata Ward^{+}, e abbiamo dimostrato che può essere vista come un insieme di Dyadic TGDs.

Infine, a supporto dei risultati teorici ottenuti nella tesi, abbiamo implementato un algoritmo per la valutazione di query su classi Ward^{+}, che sfrutta il sistema sviluppato per Shy.

Contents

Introduction

The aim of this chapter is to introduce the context and describe the main motivations that inspired this PhD thesis. In particular, we give a summary of the main results about Ontology-Based Query Answering, according to the existing literature. We close with a structural outline of the thesis.

Context and State-of-the-art

Ontology-Based Query Answering (OBQA) consists in querying data–bases by taking ontological knowledge into consideration. This is a very interesting research topic studied in database theory [5, 25, 26, 51], artificial intelligence [10, 16, 34] and in logic [6, 18].

On top of that, OBQA is stringently affiliated to several other important areas such as data integration [62], data exchange [12], and consistent query answering [67, 68]. In particular, the goal of OBQA is to find certain answers to q, i.e., the query has to be true in every possible model of the theory [7, 57].

A well-known query language based on the logic programming paradigm and that has been used for over four decades is Datalog, for which we refer the reader to papers [1, 37]. In short, a basic Datalog program comprises of a set of universally quantified variables. In the process of writing a Datalog program, just as it usually is in logic programming, we take into consideration some sets of rules to be conjunctions, the use of commas to conjoin atoms, and the assumption of all variables to be universally quantified, while excluding the universal quantifiers. The predicate symbols which appear in a program of such sort refer either to *extensional database (EDB) predicates* which values are given through an input database, or to *intensional database (IDB) predicates* which have their values computed by the program. EDB predicate symbols can appear only in rule bodies in standard Datalog. However, to consider an EDB D and a Datalog ontology Σ, the logical theory which contains both the rules of Σ and the ground atoms (called facts) of D is denoted by $D \cup \Sigma$.

However, for the purpose of ontology querying, an extension has been added to Datalog which permits the expression of the existence of certain values, which do not need to be part of the active domain (i.e., every value that comes up as argument of EDB facts or are definitely cited in the Datalog ontology) of the EDB. This extension is achieved by allowing existentially quantified variables in rule heads [72]. However, during the years, several other variants of Datalog have been introduced in the literature. Basically, these variants can be obtained

- permitting characteristics like the equality predicate, existential quantifiers and the truth constant false to appear in rule heads;

- limiting the syntactical conditions of the emerging language, so as to accomplish decidability and in applicable cases, even tractability.

The family of all such (existing and future) variants is called Datalog$^{\pm}$ [30].

This body of work focuses on ontologies which are expressed through existential rules which are also regarded as *tuple generating dependencies (TGDs)* or Datalog$^{\exists}$ rules. However, for sets consisting of TGDs alone, a large number of basic reasoning and query answering problems have the issue of undecidability. Particularly, provided a database D and a set Σ of TGDs, verifying if $D \cup \Sigma \models q$ for a ground fact q is undecidable [20]. Sadly, undecidability remains even in cases where both q and Σ are fixed, and just D is given as an input [27]. Because of this, it is necessary to identify large classes of formalisms for rule sets Σ that:

- contingent on Datalog, and therefore allow a modular rule-based style of knowledge representation;

- are syntactical fragments of first-order logic in such a way that answering a BCQ q under Σ regarding an input database D is equivalent to the classical entailment check $D \cup \Sigma \models q$;
- are sufficiently expressive in such a way that the are useful in real applications in the aforementioned areas;
- have decidable query answering;
- have good query answering complexity properties especially in the case where Σ and q are fixed. A complexity of this nature is known as *data complexity*, and is an important measure, because we can assume in a realistic sense that the EDB D is the sole really large object in the input.

It can be observed that OBQA can be reduced to the problem of answering q over a *universal model* U, i.e., a model which can be embedded homomorphically into every other model of the logical theory. A possible method which can be used to compute a universal model is to apply the so-called *chase procedure*. It uses a database D and a set Σ of constraints (for example TGDs which were introduced earlier) as input and if it terminates, it gives as result a finite instance D_Σ such that it is a universal model of D and Σ. The fundamental feature of the chase procedure is that it gives rise to a universal model which is independent of the order and precedence in which the rules are processed. Then it follows that for each *boolean conjunctive query (BCQ)* q (i.e. an existentially quantified conjunction of atoms),

$$D \cup \Sigma \models q \Leftrightarrow \mathsf{chase}(D, \Sigma) \models q.$$

However, there are cases where the universal model constructed by the chase is infinite and, as remarked above, there are cases where the problem of deciding whether a database and a set of TGDs entail a query is undecidable [41, 44]. Nonetheless, even if the chase is not finite, it remains an efficacious tool for query answering purpose, since in some cases, it is sufficient to execute the chase up to a finite level for being able to correctly answer a BCQ [34]. In the literature, five variants of the chase have been proposed: *oblivious* [27], *skolem* [69], *restricted* [44], *core* [41], and *parsimonious* [63]. Particularly, this body of work concentrates on the parsimonious variant and this is a procedure which corrects rule violations solely if the (inferred) head atom cannot be mapped homorphically to any other atom which was produced previously.

Motivations and objectives

The huge literature regarding OBQA is continuously growing, as new paradigms are being defined to ensure decidability of query answering under existential rules.

The above mentioned family of languages Datalog$^{\pm}$ is divided into the sublanguages based on the following syntactic properties: *weak acyclicity* [44], *guardedness* [27], *linearity* [29], *stickiness* [31], and *shyness* [63]. Thanks to these properties, the basic classes of existential rules have been defined and they are called: weakly acyclic, (weakly) guarded, linear, sticky(−join), and shy. Moreover, a lot of variants and combinations of these classes have been determined and studied in [17, 33, 40, 47, 61]. Lastly, decidable abstract classes of Datalog$^{\exists}$ programs exist called FES, BTS and FUS and these depend on the semantic properties [16] which include the classes named above. For more details we refer to Section 2.6.

However, also if several paradigms have been defined, in most cases each of them requires the development of a solver and, only rarely, the definition of a new class allows the use of existing systems. This gap between the number of existing paradigms and the number of developed tools, prompted us to define a combination of Shy and Ward, two classes well-know in the literature, with the aim of exploiting the tool developed for Shy[1]. In particular, we have chosen to focus on these two classes since both enjoy good properties, like including Datalog, have a good compromise between expressivity and complexity and, as aforementioned, have an implementation.

Studying how to merge these classes, we have realized that it would be possible to extend our idea (i.e., combine and generalize existing classes with the possibility to make the most of existing systems), in

[1]There exists also the solver for warded, Vadalog, but it is not available for research use.

a more general way. For that reason, we have defined the class of *Dyadic TGDs*, that allows to extend in a uniform and elegant way all the decidable classes, while using the existent related systems.

At the same time, we have defined also a combination of Shy and Ward, named Ward^+ that can be seen as a Dyadic set of TGDs.

Contributions

The first contribution of this thesis is the definition of this new decidable paradigm for ontological query answering, called Dyadic TGDs. As explained above, thanks to this new class it is possible to extend and add some expressive power to existing decidable classes. In particular, for a decidable class $\mathcal{C}$, Dyadic-$\mathcal{C}$ is the class of all the sets Σ of TGDs that admits a *dyadic decomposition* $(\Sigma_{\mathrm{HG}}, \Sigma_{\mathcal{C}})$ w.r.t. $\mathcal{C}$ (i.e., there exists a rewriting of Σ in an equivalent set of TGDs given by $\Sigma_{\mathrm{HG}} \cup \Sigma_{\mathcal{C}}$, where Σ_{HG} is a set of *head-ground* rule and $\Sigma_{\mathcal{C}} \in \mathcal{C}$). We show that the class Dyadic-$\mathcal{C}$ is decidable, providing a sound and complete algorithm used to complete the database with all the ground atoms that is possible to derive from the component Σ_{HG} of the dyadic decomposition, in order to exploit only the component $\Sigma_{\mathcal{C}}$ for query answering purposes.

The second contribution consists in the definition of a new class of $\mathrm{Datalog}^{\exists}$ program, called Ward^+, derived from the combination of two existing classes: Shy and Ward, since both of them enjoy good properties, like including Datalog, reaching a good compromise between expressibility and complexity, and, last but not least, have an implementation. We are going to prove the complexity of this new fragment with the support of two different techniques:

(i) exploiting the Dyadic class and proving that $\mathsf{Ward}^+ \subset$ Dyadic-Shy;

(ii) taking advantage of the concept of "well-behaved" *proof tree* (for more details, refer to [23] or Section 2.3), which enables the development of a Datalog rewriting for Ward^+in the future.

Finally, we implement an algorithm for the class Ward^+, based on the existence of a dyadic decomposition for Ward^+ sets of TGDs.

Structure of the Thesis

This body of work is subdivided in seven chapters:

- In Chapter 1 we provide a formal definition of the syntax and semantics of Ontology Based Query Answering.

- In Chapter 2 we give the preliminaries for this work of thesis. In Section 2.1 and in Section 2.2we describe a variant of the chase procedure, the so-called *parsimonious chase* (pchase), that has been introduced by Leone et al. in [63], and present some results contained in the work of Amendola et al. [8]. In Section 2.3 we describe the notion of *proof tree* and in Section 2.4 we recall the class *Ward*, both introduced in the work of Berger et al. [23]. Finally, we compare the classes Shy and Ward in Section 2.5, and we conclude the this chapter with an overview of other existing decidable classes in Section 2.6.

- In Chapter 3 we present one of the main theoretical contribution of this thesis. We define a new decidable paradigm for ontological query answering, called *Dyadic TGDs*, in order to provide a way to extend and add some expressive power to existent decidable classes.

- In Chapter 4 we present the second main result of this thesis: the new class Ward$^+$. This new fragment of Datalog$^\pm$ has been inspired from the combination of two existing classes: Shy and Ward.

- In Chapter 5 we present the architecture of the prototype system regarding the class Ward$^+$., followed by the experimental results.

- In Chapter 6 we present the conclusion of our work.

CHAPTER 1

Ontology Based Query Answering

In this chapter, we give a formal definition of the syntax and semantics of Ontology Based Query Answering. For further details see [1].

1.1. Basics

In this section we recall some well know notions that will be used in all the thesis.
Let $\Delta = \Delta_C \cup \Delta_N \cup \Delta_V$ the domain of the *terms*, consisting of the union of the three countably infinite domains of *constants, nulls* and *variables*, respectively. We write φ to denote a null; x a variable; $\underline{a}$ an atom, that is an expression of the form $P(\mathbf{t})$, where $P = \mathsf{pred}(\underline{a})$ is a predicate, $\mathbf{t} = t_1, \ldots, t_k$ is a tuple of *terms*, $k = \mathsf{arity}(\underline{a})$ is the arity of $\underline{a}$ or P, and a *position* $P[i]$, where $i \in \{1, \ldots, k\}$, identifies the i-th term of $\underline{a}$. For brevity we may write $[k]$ for the set $\{1, \ldots, k\}$, where $k \geq 0$. Moreover, $\mathsf{const}(\underline{a})$ ($\mathsf{vars}(\underline{a})$, resp.) is the set of constants (variables, resp.) occurring in $\underline{a}$. A *fact* is an atom that contains only constants. The set of the predicates is denoted by $\mathcal{R}$. A *substitution* is a total mapping $s : \Delta \to \Delta$. The restriction of s to a subset T of Δ, denoted by $s_{|T}$, is the substitution $\{t \mapsto s(t) \mid t \in T\}$. Let χ_1 and χ_2 be two structures containing atoms. An *homomorphism* $h : \chi_1 \to \chi_2$ is a substitution such that: (i) if $c \in \Delta_C$, then $h(c) = c$; (ii) if $p(t_1, \ldots, t_n) \in \chi_1$, then $h(p(t_1, \ldots, t_n)) = p(h(t_1), \ldots, h(t_n)) \in \chi_2$. We write $h(\chi_1)$ for the set of atoms $\{h(\underline{a}) \mid \underline{a} \in \chi_1\}$. An *isomorphism* between two atoms is a bijective homomorphism, i.e. a one-to-one correspondence between two atoms.

1.2. Syntax

Relational Databases

A *schema* $\mathbf{S}$ is a finite set of relation symbols (or predicates). The set of positions of a schema $\mathbf{S}$, denoted $\mathsf{pos}(\mathbf{S})$, is defined as $\{P[i] \mid P \in$

$S \wedge \mathsf{arity}(P) = n,$ with $n \geq 1 \wedge i \in [n]\}$. An *instance* over $\mathbf{S}$ is a (possibly infinite) set of atoms over $\mathbf{S}$ that contains constants and nulls, while a *database* D over $\mathbf{S}$ is a finite set of facts over $\mathbf{S}$. The *active domain* of an instance I, denoted $\mathsf{dom}(I)$, is the set of all terms occurring in I, while the *Herbrand Base* of I, denoted $\mathrm{HB}(I)$, is the set of all atoms that can be formed using the predicate symbols of $\mathbf{S}$ and arguments in $\mathsf{dom}(I)$. We observe that this is an extension of the classical notion of Herbrand Base, which includes ground atoms only.

Conjunctive Queries

A *conjunctive query* (CQ) over $\mathbf{S}$ is a first-order formula of the form

$$q(\mathbf{x}) := \exists \mathbf{y}(P_1(\mathbf{z_1}) \wedge \cdots \wedge P_n(\mathbf{z_n})),$$

where each $P_i(\mathbf{z_i})$, for $i \in [n]$, is an atom without nulls over $\mathbf{S}$, each variables mentioned in the $\mathbf{z_i}$'s appears either in $\mathbf{x}$ or $\mathbf{y}$, and $\mathbf{x}$ are the *output variables* of q. For convenience, we adopt the the rule-based syntax of CQs, i.e., a CQ as the one above will be written as the rule

$$Q(\mathbf{x}) \leftarrow (P_1(\mathbf{z_1}), \ldots, P_n(\mathbf{z_n})),$$

where Q is a predicate used only in the head of the CQs. A *Boolean conjunctive query* (BCQ) is a CQ of arity zero.
Let $\mathsf{atoms}(q) = \{P_1(\mathbf{z_1}), \ldots, P_n(\mathbf{z_n})\}$. The *evaluation* of $q(\mathbf{x})$ over an instance I, denoted $q(I)$, is the set of all tuples $h(\mathbf{x})$ of constants, where h is a homomorphism from $\mathsf{atoms}(q)$ to I, such that $h(\mathsf{atoms}(q)) \subseteq I$.

Tuple-Generating Dependencies

A Tuple Generetic Dependences (TGDs) is a formula of the form

$$\forall \mathbf{x} \Phi(\mathbf{x}) \rightarrow \exists \mathbf{y} \Psi(\mathbf{x}, \mathbf{y})$$

where $\mathbf{x}, \mathbf{y}$ are tuples of variables of Δ_V, and Φ, Ψ are conjunctions of atoms without constants and nulls. For brevity, we omit the universal quantifier (we write σ as $\Phi(\mathbf{x}) \rightarrow \exists \mathbf{y} \Psi(\mathbf{x}, \mathbf{y})$).

We refer to Φ and Ψ as the body and head of σ, denoted $\mathsf{body}(\sigma)$ and $\mathsf{head}(\sigma)$, respectively. If $\mathsf{head}(\sigma)$ contains only one atom, we say that

σ is a *single-head* TGD; otherwise σ is a *multi-head* TGD. Moreover, given a set Σ of TGDs, $\mathsf{hp}(\Sigma)$ represent the set of all the predicates in any head of Σ and $\mathsf{bp}(\Sigma)$ the set of all the predicates in any body of Σ.

The *frontier* of the TGD σ, denoted $\mathsf{front}(\sigma)$, is the set of variables that appear both in the body and the head of σ. We also write $\mathsf{var}_\exists(\sigma)$ for the existentially quantified variables of σ. The schema of a set Σ of TGDs, denoted $\mathsf{sch}(\Sigma)$, is the set of predicates in Σ, while $\mathsf{arity}(\Sigma) = \max\limits_{P \in \mathsf{sch}(\Sigma)} \mathsf{arity}(P)$. Furthermore, for simplicity of exposition, we write $\mathsf{pos}(\Sigma)$ instead of $\mathsf{pos}(\mathsf{sch}(\Sigma))$. In the sequel we use the notation $\mathsf{const}(\Sigma)$ ($\mathsf{var}_\exists(\Sigma)$, respectively) to extend the set above defined to the whole set Σ.

By abuse of notation, we may treat a tuple of variables as a set of variables, and a conjunction of atoms as a set of atoms. An instance I satisfies a TGD σ as the one above, written $I \models \sigma$, if the following holds: whenever there exists a homomorphism h such that $h(\Phi(\mathbf{x})) \subseteq I$, then there exists $h' \supseteq h_{|\mathbf{x}}$ such that $h'(\Psi(\mathbf{x}, \mathbf{y})) \subseteq I$. The instance I satisfies a set Σ of TGDs, written $I \models \Sigma$, if $I \models \sigma$ for each $\sigma \in \Sigma$. Finally, from now on, we assume that for each pair $\langle \sigma_1, \sigma_2 \rangle$ of rules of Σ, $\mathsf{vars}(\sigma_1) \cap \mathsf{vars}(\sigma_2) = \emptyset$.

1.3. Semantics

Models

The main reasoning task under TGD-based languages is *conjunctive query answering*. Given a database D and a set Σ of TGDs, a model of D and Σ is an instance I such that$I \supseteq D$ and $I \models \Sigma$. Let $\mathsf{mods}(D, \Sigma)$ be the set of all models of D and Σ. The certain answers to a CQ q w.r.t. D and Σ are defined as the set tuples

$$\mathsf{cert}(q, D, \Sigma) := \bigcap_{I \in \mathsf{mods}(D, \Sigma)} q(I).$$

Finally, we introduce the notion of *universal model*, that is related to the solution of the CQAns problem. A model U of D and Σ is called *universal* if, for each $M \in \mathsf{mods}(D, \Sigma)$, there is a homomorphism from U to M.

Query Answering

The answer to a BCQ q w.r.t. D and Σ is *positive*, denoted $D \cup \Sigma \models q$, if $\langle\rangle \in \mathsf{cert}(q, D, \Sigma)$. The problem of computing certain answers for a CQ w.r.t. a database and a set of TGDs from a certain class $\mathcal{C}$, is called *CQ answering*, and it is defined as follows:

PROBLEM: $\mathsf{CQAns}(\mathcal{C})$
INPUT: A database D, a set $\Sigma \in \mathcal{C}$ of TGDs, a CQ $q(\mathbf{x})$, and a tuple $\mathbf{c} \in \mathsf{dom}(D)^{|\mathbf{x}|}$.
QUESTION: Is it the case that $\mathbf{c} \in \mathsf{cert}(q, D, \Sigma)$?

In case that q is a BCQ, the above problem is called *BCQ answering*, and denoted by $\mathsf{BCQAns}(\mathcal{C})$.

Moreover, we recall that problems $\mathsf{BCQAns}(\mathcal{C})$ and $\mathsf{CQAns}(\mathcal{C})$ are logspace-equivalent [38], and we will refer to both of them.

Moreover, computing the complexity of these problem, we can refer to:

- the *data complexity* of the problem, which measures the complexity assuming that the set of TGDs $\Sigma \in \mathcal{C}$ and the CQ q are fixed and, hence, it is calculated taking only the database D as input;
- the *combined complexity* of the problem, that is calculated considering as input also the query q, the set of TGDs Σ and the database D.

1.4. The Chase and Its Variants

In this section we present one of the main algorithm in database theory: the *chase procedure* [42, 58, 65, 66]. It has several important uses, such as query equivalence and query optimization [1], containment of queries under constraint [2], computing data exchange solutions [44], checking logical implication of constraints [19, 65], and query answering under constraints [27].

This procedure takes as input a database D and a set Σ of constraints (such as TGDs previously introduced) and, if it terminates, its result is a finite instance D_Σ such that it is a model of D and Σ, and in particular, it is a universal model, i.e., it can be homomorphically mapped to any other model of D and Σ. We denote the result of the chase for D under Σ with $\mathsf{chase}(D,\Sigma)$. The key property of the chase procedure is that it produces an universal model independently of the order in which rules are processed. Then, it follows that, for each BCQ q,

$$D \cup \Sigma \models q \Leftrightarrow \mathsf{chase}(D \cup \Sigma) \models q.$$

However, there are cases where the universal model found by the chase is infinite and also there are cases where the problem of deciding whether a database and a set of TGDs entail a query is undecidable [41, 44]. Nevertheless, even if the chase is not finite, it remains an efficacious tool for query answering purpose, since in some cases, it is sufficient to execute the chase up to a finite level for being able to answer a BCQ [34]. Roughly speaking, the chase "repairs" a database w.r.t a set of dependencies so that the resulted database satisfies the dependencies. It starts from an instance and exhaustively performs a sequence of rule applications with respect to a fixed criterion, which depends on the considered chase variant.
In literature, five variants of the chase have been proposed: *oblivious* [27], *skolem* [69], *restricted* [44], *core* [41], and *parsimonious* [63]. To understand the differences among these variants, we need to introduce the notion of *firing homomorphism* for a rule σ and an instance I.

DEFINITION 1.1 ([64]). A **firing homomorphism** for the pair $\langle \sigma, I \rangle$ is any homomorphism $h : \mathsf{body}(\sigma) \to I$ s.t. $h = h|_{\mathsf{vars}(\mathsf{body}(\sigma))}$. The *fire* of σ via h produces the atom $\mathit{fire}(\sigma, h)$, obtained from $h(\mathsf{head}(\sigma))$ by replacing each existential variable of σ with a different fresh null, i.e., not already occurring in I. After that, h is said to be *spent* with respect to σ. The result of spending σ over I with h is the instance $I' = I \cup \mathit{fire}(\sigma, h)$. Such a single step is denoted $I\langle \sigma, h \rangle I'$.

Depending on the variant of the chase under consideration, the fire of σ via h may be subject to a specific fire condition. We recall the definition of the oblivious and restricted chase, useful for our purpose. For the definition of the parsimonoius variant, we refer the reader to Section

2.1. Consider any instance $I' \supseteq I$. In the case of the restricted chase, we say that the pair $\langle \sigma, h \rangle$ satisfies the fire condition with respect to I' if there is no homomorphism $h' \supseteq h$ from $\{\mathsf{head}(\sigma)\}$ to I'. In contrast, in the case of the oblivious chase (ochase or chase), the pair $\langle \sigma, h \rangle$ always satisfies the fire condition with respect to I'. We remark that we assume that nulls introduced at each fire functionally depend on the pair $\langle \sigma, h \rangle$ that is involved in the fire. The last assumption has the immediate consequence that (regardless of the order in which the rules and the firing homomorphisms are processed) $\mathsf{ochase}(D \cup \Sigma)$ can be considered unique (up to isomorphism) and that $\mathsf{rchase}(D \cup \Sigma) \subseteq \mathsf{ochase}(D \cup \Sigma)$.

Hence, as stated above, the main idea of the chase is, starting from a database D, to exhaustively apply, in a "fair" way, firing homomorphisms for the given set Σ of TGDs on the instance constructed so far. More formally, a *chase sequence for D under Σ* is a (possibly infinite) sequence $(I_i \langle \sigma_i, h_i \rangle I_{i+1})_{i \geq 0}$ of chase steps such that: (i) $I = I_0$; (ii) for each $i \geq 0$, $\sigma_i \in \Sigma$; and (iii) $\bigcup_{i \geq 0} I_i \models \Sigma$. We call $\bigcup_{i \geq 0} I_i$ the *result* of this chase sequence, which always exists. Finally, we recall that $\mathsf{chase}(D \cup \Sigma)$ can be decomposed into *levels* [31]: D has level 0, and an atom has level $\gamma + 1$ if it is obtained, during the chase, due to atoms with maximum level γ. We refer to the part of the chase up to level γ as $\mathsf{chase}^{\gamma}(D \cup \Sigma)$. The limit of $\mathsf{chase}^{\gamma}(D \cup \Sigma)$ for $\gamma \to \infty$ is $\mathsf{chase}(D \cup \Sigma)$.

CHAPTER 2

State of the Art

In this chapter, we report some background material essential for the thesis. In Section 2.1 and in Section 2.2 we describe the variant of the chase named Parsimonious Chase and the class *Shy* of TGDs introduced in [63]. In Section 2.3 we describe the notion of proof tree that leads to space-bounded algorithms, and in Section 2.4 we recall the class *Ward*, both introduced in the work of Berger et al. [23]. Successively, we compare the classes Shy and Ward in Section 2.5. Finally, in Section 2.6, we conclude the chapter with an overview of other existing decidable classes.

2.1. The Parsimonious Chase Algorithm

In this section we present a variant of the chase procedure, the so-called parsimonious chase (pchase) [63, 64], that has been introduced in the last years. This variant allows to define two Datalog$^{\exists}$ classes: *Parsimonious* and *Shy* (for the last one see Section 2.2).

Intuitively, the pchase procedure repairs violations of rules only if the (inferred) head atom cannot be homomorphically mapped to any atom previously produced. Moreover, it is sound and complete for some classes, its termination is always guaranteed, and computational complexity has been studied (see [63, 64]).

2.1.1. Formal Definition. Now, we formally define a variant of the chase procedure, the *parsimonious chase* (pchase), and the class of Datalog$^{\exists}$ programs that derives from it. As discussed in Section 1.4, the chase of a database D in the presence of a set Σ of TGDs is a procedure that constructs a universal model for $D \cup \Sigma$. However, this procedure does not always terminate. One of the main properties of the pchase is that it terminates on any Datalog$^{\exists}$ program.

Before defining the pchase procedure, for the reader convenience, we recall the definition of firing homomorphism given in Section 1.4.

Definition 1.1. Consider a rule σ and an instance I. A **firing homomorphism** for the pair $\langle\sigma, I\rangle$ is any homomorphism $h : \mathsf{body}(\sigma) \to I$ s.t. $h = h|_{\mathsf{vars}(\mathsf{body}(\sigma))}$. The *fire* of σ via h produces the atom *fire*(σ, h), obtained from $h(\mathsf{head}(\sigma))$ by replacing each existential variable of σ with a different fresh null, i.e., not already occurring in I. After that, h is said to be *spent* with respect to σ .

We know that the fire condition is not unique, since it depends on the variant of the chase under consideration. As we focus on the pchase, in the following we give the definition of *parsimonious fire condition*. Intuitively, the parsimonious fire condition prevents from producing atoms that could be homomorphically mapped to any atom previously produced. More formally, we define the parsimonious fire condition as follows.

Definition 2.1. Consider a rule σ, an instance I, and a firing homomorphism h for $\langle\sigma, I\rangle$. The pair $\langle\sigma, h\rangle$ satisfies the **parsimonious fire condition** with respect to an instance $I' \supseteq I$ if there is no homomorphism h' from $\{h(\mathsf{head}(\sigma))\}$ to I' such that $h'(x) = h(x)$, for each $x \in \mathsf{front}(\sigma)$ such that $h(x) \in \Delta_C$.

The **parsimonious chase** (denoted pchase) is obtained by the following procedure.

Algorithm 1 ([64]). The parsimonious chase procedure.

Input: A Datalog$^\exists$ set of TGDs Σ, and a database D.
Output: $\mathsf{pchase}(D \cup \Sigma)$.

```
(1) $I := D$;
(2) **foreach** $\sigma \in \Sigma$ **do**
(3)       **foreach** unspent (w.r.t. $\sigma$) firing homomorphism $h$ for the pair $\langle\sigma, I\rangle$ **do**
(4)             **if** $\langle\sigma, h\rangle$ satisfies the parsimonious fire condition w.r.t. $D$ **then**
```

```
(5)                 $D = D \cup \{\mathit{fire}(\sigma, h)\}$;
(6)  if $I \neq D$ then
(7)       goto Step 1;
(8)  else return $I$;
```

Now we compare, with the aid of the following example, the behavior of the parsimonious chase with the oblivious one.

Example 2.2. Consider the database $D = \{P(a)\}$, and a set Σ consisting of the following rules.

$$\begin{array}{rrcl} \sigma_1 : & P(x_1) & \rightarrow & \exists\, y_1\ Q(x_1, y_1) \\ \sigma_2 : & Q(x_2, y_2) & \rightarrow & P(y_2) \end{array}$$

The fire of rule σ_1 via the homomorphism $\{x_1 \mapsto a,\ y_1 \mapsto \varphi_1\}$ produces the atom $Q(a, \varphi_1)$. However, according to Definition 2.1, the pair $\langle \sigma_2, \{x_2 \mapsto a, y_2 \mapsto \varphi_1\}\rangle$ does not satisfies the parsimonious fire condition, since we would have produced the atom $P(\varphi_1)$, which would have been homomorphically mapped to $P(a)$. Hence, the parsimonious chase terminates after a finite number of steps, and

$$\mathsf{pchase}(D \cup \Sigma) = \{P(a), Q(a, \varphi_1)\}.$$

On the other hand, considering the oblivious chase (see Section 1.4), we do not obtain a finite set, but an infinite one, given by:

$$\mathsf{ochase}(D \cup \Sigma) = \mathsf{pchase}(D \cup \Sigma) \cup \{P(\varphi_i)\}_{i \in \mathbb{N}^+} \cup \{Q(\varphi_i, \varphi_{i+1})\}_{i \in \mathbb{N}^+}.$$

We point out that, differently to $\mathsf{ochase}(D \cup \Sigma)$, the $\mathsf{pchase}(D \cup \Sigma)$ might not be a model of $D \cup \Sigma$. Now we define the class of TGDs depending on the semantic property called *parsimony*.

Definition 2.3. (*Parsimony*) Consider a database D and a set Σ of TGDs. We say that Σ is **parsimonious** if, for each atom $\underline{a}$ of $\mathsf{ochase}(D \cup \Sigma)$, there exists a homomorphism from $\underline{a}$ to $\mathsf{pchase}(D \cup \Sigma)$.

Let $\mathsf{Parsimonious}$ denote the class of all parsimonious sets of TGDs.

Considering Example 2.2, the set Σ is parsimonious. In fact, having in mind that

$$\mathsf{pchase}(D \cup \Sigma) = \{P(a), Q(a, \varphi_1)\}, \text{and}$$

$$\mathsf{ochase}(D \cup \Sigma) = \mathsf{pchase}(D \cup \Sigma) \cup \{P(\varphi_i)\}_{i \in \mathbb{N}^+} \cup \{Q(\varphi_i, \varphi_{i+1})\}_{i \in \mathbb{N}^+},$$

it is clear that for each atom of the form $P(\varphi_i)$ there exists a homomorphism

$$h = \{\varphi_i \mapsto a\} \text{ such that } h(P(\varphi_i)) \in \mathsf{pchase}(D \cup \Sigma),$$

and for each atom of the form $Q(\varphi_i, \varphi_{i+1})$ there exists a homomorphism

$$h = \{\varphi_i \mapsto a, \varphi_{i+1} \mapsto \varphi_1\} \text{ such that } h(Q(\varphi_i, \varphi_{i+1})) \in \mathsf{pchase}(D \cup \Sigma).$$

Moreover, from Example 2.2, we catch sight of one of the main properties of the parsimonious chase, that is, it *always terminates*. This variant of the chase enjoys different good properties, that we group together in the next section. For more details, see [63, 64].

2.1.2. Main properties. In this section we list some of the principal characteristics of the parsimonious chase [63, 64], providing some new results obtained in [8].

We highlight a main property of the pchase, based on isomorphic atoms, a crucial notion in several Datalog$^{\pm}$ classes [28].

PROPOSITION 2.4. *Consider a database D and a set Σ of TGDs. Then, $\mathsf{pchase}(D \cup \Sigma)$ does not contain isomorphic atoms.*

PROOF. Assume towards a contradiction, that there are two isomorphic atoms $\underline{a}$ and $\underline{a}'$ in $\mathsf{pchase}(D \cup \Sigma)$. Thus, there is a homomorphism h from $\{\underline{a}\}$ to $\{\underline{a}'\}$ s.t. h^{-1} is a homomorphism from $\{\underline{a}'\}$ to $\{\underline{a}\}$. Without loss of generality, assume that $\underline{a} \in I$, for some I generated during the pchase procedure. As $\underline{a}' \in \mathsf{pchase}(D \cup \Sigma)$, then there is a rule ρ, an instance $I' \supseteq I$, and an unspent firing homomorphism h' for $\langle \rho, I' \rangle$, s.t. $\mathit{fire}(\rho, h') = \underline{a}'$, against the fact that $h^{-1} \circ \sigma$ is a homomorphism from $\{h'(\mathsf{head}(\rho))\}$ to I'. Indeed, $(h^{-1} \circ \sigma)(h'(\mathsf{head}(\rho)) = h^{-1}(\sigma(h'(\mathsf{head}(\rho)) = h^{-1}(\mathit{fire}(\rho, h')) = h^{-1}(\underline{a}') = \underline{a} \in I \subseteq I'$. □

As discussed above, the principal property of the pchase is given by the following result.

Proposition 2.5 ([64]). *The parsimonious chase always terminate.*

Exploiting the proposition above, it is possible to prove that parsimony guarantees decidability of Boolean atomic query aswering.

Theorem 2.6 ([64]). *The problem* BCQAns *over parsimonious programs and atomic queries is decidable.*

Nevertheless, parsimony alone is not enough to also guarantee the decidability of Boolean conjunctive query evaluation and, therefore, of conjunctive query answering. In particular, in [64] has been proved that checking whether a program is parsimonious is not decidable, and, in particular, it is coRE-complete. This is stated in the following theorem.

Theorem 2.7 ([64]). *The problem* BCQAns *over parsimonious programs is undecidable.*

To gain decidability also for conjunctive queries, we recall a technique called *parsimonious-chase resumption*, which is sound for any Datalog$^\exists$ program, and also complete over a class named *Shy*, that we introduce next (see Section 2.2). Intuitively, the idea is to "reapply" the parsimonious chase a linear number of times in the size of the query. To better understand the intuition, consider the following example.

Example 2.8. Let D be a database and Σ a set of TGDs. Suppose that

$$\mathsf{pchase}(D \cup \Sigma) = \{P(c,\varphi), Q(d,e), R(c,e)\},$$

and that

$$\mathsf{ochase}(D \cup \Sigma) = \mathsf{pchase}(D \cup \Sigma) \cup Q(\varphi,e).$$

We point out that, according to Definition 2.1, the atom $\underline{a} = Q(\varphi,e)$ cannot belong to $\mathsf{pchase}(D \cup \Sigma)$ due to atom $\underline{b} = Q(d,e)$, since there exist an homomorphism from $\underline{a}$ to $\underline{b}$. Now, consider the Boolean conjunctive query

$$q : \exists\, x\; \exists\, y\; \exists\, z\;\, P(x,y), Q(y,z).$$

Evidently, $\mathsf{pchase}(D \cup \Sigma)$ does not provide any answer to q even if $D \cup \Sigma \models q$. To solve this issue, the idea presented in [63, 64] is to "promote" φ to a constant in Δ_C and "resume" the parsimonious chase execution at Step (2) (see Algorithm 1), in the same state in which it had stopped after returning the set I at Step (6). Now, since φ can be considered as a constant, then there is no homomorphism from $\{Q(\varphi, e)\}$ to $\mathsf{pchase}(D \cup \Sigma)$, and then the atom $Q(\varphi, e)$ can be inferred by the algorithm. In this way, $\mathsf{pchase}(D \cup \Sigma)$ provides an answer to q.

The process described above of promoting a null from Δ_N to an extra constant not already occurring in Δ_C is called *freezing*. In particular, we observe that two nulls cannot be promoted to the same constant. With the following definition, we formalize the notion of parsimonious-chase resumption.

Definition 2.9. (see Definition 4.8 in [64]) Let D be a database and Σ a set of TGDs. The output of the parsimonious chase after $k \geq 0$ resumptions is defined as follows:

(1) $\mathsf{pchase}(D \cup \Sigma, 0) = D$;
(2) $\mathsf{pchase}(D \cup \Sigma, k) = \mathsf{pchase}(\lceil \mathsf{pchase}(k-1) \rfloor \cup \Sigma)$,

where $\lceil \mathsf{pchase}(k-1) \rfloor$ is the set obtained from $\mathsf{pchase}(D \cup \Sigma, k-1)$ after freezing all of its nulls.

We observe that $\mathsf{pchase}(D \cup \Sigma, 1) = \mathsf{pchase}(D \cup \Sigma)$. Moreover, it holds that the sequence $\{\mathsf{pchase}(D \cup \Sigma, k)\}_{k \in \mathbb{N}}$ is increasing, and the limit of this sequence is denoted by $\mathsf{pchase}(D \cup \Sigma, \infty)$. The next proposition states that the resumption technique above described is sound w.r.t. QA, and that its infinite application, over particular sets of TGDs (see Section 2.5), also ensures completeness.

Proposition 2.10 (see Proposition 4.9 in [64]). *Given a database D and a set Σ of TGDs, it holds that* $\mathsf{pchase}(D \cup \Sigma, \infty) \subseteq \mathsf{ochase}(D \cup \Sigma)$.

Our next claim is to understand the nature of the pchase procedure, but first, we recall the last result useful to our purposes, obtained in [63, 64].

THEOREM 2.11 (see Theorem 6.1 in [64]). *Problem* BCQAns *for atomic queries over parsimonious programs is in* EXPTIME *in combined complexity and in* PTIME *in data complexity.*

In what follows we investigate some aspects of the Parsimonious Chase. As discussed previously, one of the principal properties of this variant is that its termination is always guaranteed. However, no precise bound has been provided so far. To this end, we decide to examine the atoms generated in the pchase, and we exploit the notion of Bell numbers, used to count the number of distinct partitions of a finite set, to compute an upper bound for the number of atoms generated by the pchase procedure. This is one of the results contained in the conference paper [8].

We aim at recognizing what kind of atoms belong to the pchase, that is strictly related to count the number of atoms generated by the pchase. To provide an exact upper bound for the pchase, we exploit the notion of "equality type" defined in [50] introducing the equivalent concept of *type*; then, we show its strictly relation to the form of non-isomorphic atoms of a given predicate.

DEFINITION 2.12. Let m be a positive integer, S an arbitrary partition of $\{1,\dots,m\}$, C a set with $|C| \leq |S|$, and $f : C \to S$ an injective map. We define the **type** of S, C and f as the family of sets

$$\mathcal{T}(S,C,f) = \Big\{ s \cup f^{-1}(s) \mid s \in S \Big\}.$$

The following example is given to better understand the previous definition.

EXAMPLE 2.13. Let $m = 6$, $C = \{c_1, c_2\}$, and let $S = \Big\{\{1,2\},\{3,6\},\{4\},\{5\}\Big\}$ be a partition of $\{1,\dots,6\}$. Consider the injective map $f : C \to S$ such that $f(c_1) = \{3,6\}$ and $f(c_2) = \{5\}$. Then,

$$\mathcal{T}(S,C,f) = \Big\{\{1,2\},\{3,6,c_1\},\{4\},\{5,c_2\}\Big\}.$$

Fixing an integer m, the next step is to count the number of all possible types that can be generated from any partition of the set $\{1,\dots,m\}$, varying C on a superset D of a fixed size d. In order to do this, we resort to the Bell number B_n, that is the number of ways to partition a set of n labeled elements.

THEOREM 2.14. *Let $m \in \mathbb{N}$ and D a finite set of size $d > 0$. The number of all possible types generated from all the partitions of the set $\{1,\dots,m\}$ and all subsets of D is given by*

$$\Upsilon_m^d = \sum_{s=1}^{m} S(m,s) \cdot \sum_{c=0}^{\min\{s,d\}} \binom{d}{c} \cdot \frac{s!}{(s-c)!},$$

where $S(m,s)$ is the Stirling number counting the number of partitions of size s on m elements.

PROOF. To prove the theorem, we first recall that, given two sets A and B with $|A| = \alpha \leq \beta = |B|$, the number of injective maps from A to B is $\frac{\beta!}{(\beta-\alpha)!}$.

Then, fixing a partition S of $\{1,...,m\}$ with $|S| = s$, the number of injective maps from any subset $C \subseteq D$ to S, with $|C| = c \leq s$, is $\frac{s!}{(s-c)!}$, while the number of subsets of size c is $\binom{d}{c}$. Thus, the number of all possible types for the fixed partition S is

$$\sum_{c=0}^{\min\{s,d\}} \binom{d}{c} \cdot \frac{s!}{(s-c)!}.$$

Hence, the number of types generated from all the partitions of the set $\{1,\dots,m\}$ and all subsets of D is given by

$$\sum_{s=1}^{m} S(m,s) \cdot \sum_{c=0}^{\min\{s,d\}} \binom{d}{c} \cdot \frac{s!}{(s-c)!},$$

where $S(m,s)$ is the Stirling number counting the number of partitions of size s on m elements. □

CONJECTURE 1. Let $m \in \mathbb{N}$, D a finite set of size $d > 0$, and B_n the n-th Bell number and consider

$$\gamma_m^d = \sum_{h=0}^{m} \binom{m}{h} d^h B_{m-h}.$$

Then, we claim that

$$\Upsilon_m^d = \gamma_m^d.$$

We point out that by experimental evaluation the two formulas coincide. We hope to formally prove this conjecture in future works.

Taking advantage of the notion of type, we can provide a new representation of an arbitrary atom.

Definition 2.15. Consider an atom $\underline{a} = p(\mathbf{t})$ of arity m. The **atom type** of $\underline{a}$ is defined as $T_{\underline{a}} = \mathcal{T}(S, C, f)$, where:

(1) $C = \mathsf{const}(\underline{a})$,

(2) $S = \Big\{\{n \mid \underline{a}[n] = t_i\} \mid \ i = 1, \dots, m\Big\}$, and

(3) $f : C \to S$ such that $f(c) = \{n \mid \underline{a}[n] = c\}$.

In plain words, the type of an atom $\underline{a}$ has the form

$$T_{\underline{a}} = \{\vartheta(t_1), \dots, \vartheta(t_m)\},$$

where ϑ is such that

$$\vartheta(t_i) = \begin{cases} \{n \mid n \in \{1, \dots, m\} \wedge \underline{a}[n] = t_i\} \cup \{t_i\}, & \text{if } t_i \text{ is a constant,} \\ \{n \mid n \in \{1, \dots, m\} \wedge \underline{a}[n] = t_i\}, & \text{otherwise.} \end{cases}$$

Therefore, the type of an atom is formed by the sets of positions where a term occurs, highlighting positions where constants occur.

Example 2.16. Consider $\underline{a} = p_1(\varphi_1, \varphi_3, \varphi_2, \varphi_1)$ and $\underline{b} = p_2(c, \varphi_1, d, c, \varphi_2, \varphi_2, \varphi_1)$. Then, $T_{\underline{a}} = \Big\{\{1,4\}, \{2\}, \{3\}\Big\}$ and $T_{\underline{b}} = \Big\{\{1,4,c\}, \{2,7\}, \{3,d\}, \{5,6\}\Big\}$.

Theorem 2.17. *Let $\underline{a} = p(t_1, \dots, t_k)$ and $\underline{a}' = p(t_1', \dots, t_k')$ be two atoms. Then, $\underline{a}$ and $\underline{a}'$ are isomorphic if, and only if,* $\mathsf{pred}(\underline{a}) = \mathsf{pred}(\underline{a}')$ *and* $T_{\underline{a}} = T_{\underline{a}'}$.

Proof. Let us consider two atoms $\underline{a}$ and $\underline{a}'$. If $\mathsf{pred}(\underline{a}) \neq \mathsf{pred}(\underline{a}')$ or $\mathsf{arity}(\underline{a}) \neq \mathsf{arity}(\underline{a}')$, then of course cannot exist an isomorphism between them. Hence, we can take for granted that the two atoms have same predicate and arity.

[$\Rightarrow$] Assume that there is an isomorphism between $\underline{a}$ and $\underline{a}'$, i.e., there is a homomorphism $h : \{\underline{a}\} \to \{\underline{a}'\}$ s.t. $h(t_i) = t'_i$, $i = 1, \dots, k$ and s.t. $h^{-1} : \{\underline{a}'\} \to \{\underline{a}\}$ is a homomorphism. Let $T_{\underline{a}} = \{\vartheta(t_1), \dots, \vartheta(t_k)\}$ and $T_{\underline{a}'} = \{\vartheta'(t_1), \dots, \vartheta'(t_k)\}$. We claim that $\vartheta(t_i) = \vartheta(t'_i)$, for $i = 1, \dots, k$. Assume that $\vartheta(t_i) \subseteq \vartheta(t'_i)$, and let $n \in \vartheta(t_i) \cap \mathbb{N}$, so that $\underline{a}[n] = t_i$. Therefore, we have that $t'_i = h(t_i) = h(\underline{a}[n]) = h(\underline{a})[n] = \underline{a}'[n]$. Hence, $n \in \vartheta(t'_i)$. Moreover, if $n = c$ is a constant, by definition of homomorphism, we have $c \in \vartheta(t_i) \Rightarrow t_i = c \Rightarrow t'_i = h(t_i) = t_i = c \Rightarrow c \in \vartheta(t'_i)$. Conversely, assume that $\vartheta(t_i) \supseteq \vartheta(t'_i)$, and let $n \in \vartheta(t'_i)$. Hence, $\underline{a}'[n] = t'_i$. Therefore, $t_i = h^{-1}(t'_i) = h^{-1}(\underline{a}'[n]) = h^{-1}(\underline{a}')[n] = \underline{a}[n]$. Thus, $n \in \vartheta(t_i)$.

[$\Leftarrow$] Let us assume that $\mathcal{T}_{\underline{a}} = \mathcal{T}_{\underline{a}'}$. Let $h : \{\underline{a}\} \to \{\underline{a}'\}$ be s.t. $h(t_i) = t'_i$. First, we prove that h is a homomorphism. Let $t_i = c$ be a constant. Suppose that $c \in \vartheta(t_i)$, then by assumption $c \in \vartheta(t'_i)$, hence $t'_i = c$. It remains to be shown that h is also injective. Let $t'_i = t'_j$. Then, $\vartheta(t'_i) = \vartheta(t'_j) \Rightarrow \vartheta(t_i) = \vartheta(t_j) \Rightarrow t_i = t_j$. □

After exploiting the notion of Bell numbers and counting the number of distinct partitions of a finite set, we are able to provide an upper bound for the maximum number of atoms generating by the pchase procedure.

THEOREM 2.18. *Consider a database D and a set Σ of TGDs. Let* $\mathsf{arity}(D \cup \Sigma) = w$*,* $|\mathsf{const}(D)| = d$*, and l_m the number of predicates in* $\mathsf{pred}(D \cup \Sigma)$ *of arity m. Then,*

$$|\mathsf{pchase}(D \cup \Sigma)| \leq \sum_{m=0}^{w} l_m \Upsilon_m^d.$$

PROOF. By Theorem 2.14 and Theorem 2.17, the total number of non isomorphic atoms over $\mathsf{pred}(D \cup \Sigma)$ and $\mathsf{const}(D) \cup \Delta_N$ is given by $\sum_{m=0}^{w} l_m \Upsilon_m^d$. Moreover, by Theorem 2.4, we know that $\mathsf{pchase}(D \cup \Sigma)$ does not contain isomorphic atoms. Hence, the statement follows. □

From now on, we denote with Γ_w^d the upper bound in Theorem 2.18.

To show that this bound is also tight, we introduce an *ordering* on types in order to show that there exists a family of ontologies for which the pchase can produce exactly the upper bound previously computed, so that it corresponds to the maximal number of atoms effectively generated by the pchase procedure.

Definition 2.19. Consider the atom types $T = \mathcal{T}(S,C,f)$ and $T' = \mathcal{T}(S',C',f')$. Then, T **precedes** T', if

(1) $|C| < |C'|$, or

(2) $|C| = |C'|$ and $|S| > |S'|$.

Intuitively, to build an ontology with a sequence of firing homomorphisms able to generate a pchase of size exactly Γ_w^d, there should exists a rule for each possible atom type, whenever constants are allowed in the rules. Otherwise, we need a predicate to collect all constants of the database. To better understand our idea, we give an example of such a program before to provide the formal result.

Example 2.20. Consider the database $D = \{t(c_1), t(c_2)\}$, and the set Σ of TGDs consisting of the following rules:

$$
\begin{array}{rrcl}
\sigma_1: & & \rightarrow & \exists x_1, y_1, z_1 \; P(x_1, y_1, z_1) \\
\sigma_2: & & \rightarrow & \exists x_2, y_2 \; P(x_2, x_2, y_2) \\
\sigma_3: & & \rightarrow & \exists x_3, y_3 \; P(x_3, y_3, x_3) \\
\sigma_4: & & \rightarrow & \exists x_4, y_4 \; P(x_4, y_4, y_4) \\
\sigma_5: & & \rightarrow & \exists x_5 \; P(x_5, x_5, x_5) \\
\sigma_6: & T(z_6) & \rightarrow & \exists x_6, y_6 \; P(z_6, x_6, y_6) \\
\sigma_7: & T(z_7) & \rightarrow & \exists x_7, y_7 \; P(x_7, z_7, y_7) \\
\sigma_8: & T(z_8) & \rightarrow & \exists x_8, y_8 \; P(x_8, y_8, z_8) \\
\sigma_9: & T(y_9) & \rightarrow & \exists x_9 \; P(x_9, x_9, y_9) \\
\sigma_{10}: & T(y_{10}) & \rightarrow & \exists x_{10} \; P(x_{10}, y_{10}, x_{10}) \\
\sigma_{11}: & T(y_{11}) & \rightarrow & \exists x_{11} \; P(y_{11}, x_{11}, x_{11}) \\
\sigma_{12}: & T(y_{12}), T(z_{12}) & \rightarrow & \exists x_{12} \; P(x_{12}, y_{12}, z_{12}) \\
\sigma_{13}: & T(y_{13}), T(z_{13}) & \rightarrow & \exists x_{13} \; P(y_{13}, x_{13}, z_{13}) \\
\sigma_{14}: & T(y_{14}), T(z_{14}) & \rightarrow & \exists x_{14} \; P(y_{14}, z_{14}, x_{14}) \\
\sigma_{15}: & T(x_{15}), T(y_{15}), T(z_{15}) & \rightarrow & P(x_{15}, y_{15}, z_{15}) \\
\sigma_{16}: & & \rightarrow & \exists x_{16} \; T(x_{16})
\end{array}
$$

We build $\mathsf{pchase}(D \cup \Sigma)$ starting from rule σ_1 to rule σ_{16}. For each rule $\sigma_i \in \Sigma$ in this ordering, we consider all firing homomorphisms h for σ_i, $i \in \{1, \dots, 16\}$. E.g., the rule σ_8 in red produces the atoms

$$\{P(\varphi_1, \varphi_2, c_1),\ P(\varphi_3, \varphi_4, c_2)\}.$$

Thus, the number of atoms with predicate P generated by the pchase will be $37 = \gamma_3^2$, and $|\mathsf{pchase}(D \cup \Sigma)| = 40 = \gamma_3^2 + \gamma_1^2$.

Now we can prove the following result.

THEOREM 2.21. *Let w be a positive integer, C a set of constants of size c, and Γ_w^c as above. Then, there is a database D and an ontology Σ_w s.t. $|\mathsf{pchase}(D \cup \Sigma_w)| = \Gamma_w^c$.*

PROOF. Consider two predicates P of arity w and T of arity 1. We set the database $\mathrm{D} = \{T(c) \mid c \in C\}$ and define the ontology Σ_w as follows. Given a partition $S_i = \{\Lambda_1, \dots, \Lambda_n\}$ of w, where $n = |S_i|$, we construct a rule r_i with an empty body, by adding $x_1, \dots, x_n$ existential variables so that $\Lambda_j = \{k \mid P[k] = x_j,\ k \in [w]\}$. Now, fix a rule r_i with $n > 1$ existential variables. We produce $n-1$ blocks of rules as follows. We translate j existential variables into universal ones, by adding j atoms over predicate T in the body. Hence, we construct $\binom{n}{j}$ rules. Then, we add the rules $P(x_1, \dots, x_w) = T(x_1), \dots, T(x_w)$, and $\exists x T(x)$. Finally, we remove all rules having in the head more than one repeated universal variable. To prove that $|pchase(D \cup \Sigma_w)| = \Gamma_w^d$, we provide a sequence of $\Gamma_w^d - d$ firing homomorphisms. To each rule r in Σ_w we associate uniquely an atom $g(\mathsf{head}(r))$, where g maps existential variables to fresh nulls, and universal variables to a fixed constant. The type ordering on the atoms gives an ordering on the rules, and so to the sequence of firing homomorphisms. □

We observe that the maximal number of distinct atoms that can be generated by the pchase procedure identified above, γ_m^d, improves the bound given in [63], that is $(d+m)^m$. In particular, $d^m \leq \gamma_m^d \leq (d+m)^m$. Since in the OBQA context, normally, d is much bigger than m, it could seem that the effort to find such a precise upper bound can be useless for practical purposes. However, this is not the case, since the search for a precise upper bound led to identify the fundamental notions of type and type ordering that highlighted some qualitative characteristics of the pchase. Moreover, there could be other contexts where m is much bigger than d (think for example to scenarios where tuples encode strings over a certain alphabet, as in complexity proofs based on Turing Machine simulation). In these cases, our bound represents a concrete improvement.

2.2. The Shy Class

In this section we introduce the class Shy, an easily recognizable class that enjoys the parsimony property (see Definition 2.3), and that extends both the class of Datalog (the well-know class of existential-free rules of the form $\forall \mathbf{x} \forall \mathbf{y}(\phi(\mathbf{x},\mathbf{y}) \rightarrow P(\mathbf{x}))$), and linear Datalog$^\exists$ programs (i.e., with at most one body atom), while preserving the same (data and combined) complexity of QA over Datalog, even though it includes existential quantifiers. Nevertheless, it is incomparable to the other main classes ensuring decidability. Before to start with the explanation about its syntax, for the reader convenience, we recall the definitions regarding the classification of variables, stated in the works of Leone et al. [63, 64].

Definition 2.22. (see Definition 5.1 in [64]) Consider a set Σ of TGDs, a variable $y \in \mathsf{var}_\exists(\Sigma)$, a predicate R of arity k, and an index $i \in \{1,\ldots,k\}$. A position $R[i]$ is **invaded** by y if there is a rule $\sigma \in \Sigma$ such that $\mathsf{head}(\sigma) = R(t_1,\ldots,t_k)$ and either $t_i = y$, or t_i is a $\forall$-variable which occurs in the body of σ only in positions that are invaded by y.

Definition 2.23. ([64]) Let Σ be a set of TGDs. Fix a TGD $\sigma \in \Sigma$ and a variable x in $\mathsf{body}(\sigma)$:

- x is **attacked** by a variable y if x occurs in $\mathsf{body}(\sigma)$ only in positions that are invaded by y;
- x is **protected** if it is no attacked by no variable.

Now we are able to define the class Shy.

DEFINITION 2.24. (*Shyness*, See Definition 4.2 in [63]) A set Σ of TGDs is **shy** if, for each TGD $\sigma \in \Sigma$ the following conditions are both satisfied:

(1) if a variable x occurs in more than one body atom, then x is protected in $\mathsf{body}(\sigma)$;

(2) if two distinct frontier variables are not protected in $\mathsf{body}(\sigma)$ and occur in two different body atoms, then they are not attacked by the same variable.

The class of shy sets of TGDs is denoted by Shy.

In order to better understand the previous definitions, now we give an example of a shy set of TGDs.

EXAMPLE 2.25. Consider the database $D = \{P(a)\}$, and the following set Σ of rules.

$$\begin{array}{lrcl}
\sigma_1 : & P(x_1) & \rightarrow & \exists\, y_1\ Q(x_1, y_1) \\
\sigma_2 : & Q(x_2, y_2) & \rightarrow & S(y_2) \\
\sigma_3 : & P(x_3), S(z_3) & \rightarrow & \exists\, y_3\ R(x_3, y_3) \\
\sigma_4 : & R(x_4, y_4) & \rightarrow & T(y_4) \\
\sigma_5 : & S(x_5), Q(x_5, y_5), T(x_5) & \rightarrow & V(x_5)
\end{array}$$

According to Definition 2.24, it is easy to see that rules σ_1, σ_2 and σ_4 are shy, since in their bodies there is only one atom. Rule σ_3 does not even violate the shyness condition, because this rule does not contain joins or attacked variables. However, the body of rule σ_5 is composed by three atoms: $S(x_5)$, $Q(x_5, y_5)$ and $T(x_5)$. In particular, we have that $S[1]$ is invaded by y_1, $Q[1]$ is invaded by y_1, but $T[1]$ is invaded by y_3. Hence, we have that the variable x_5 is protected. Therefore, Σ is a shy set of TGDs.

Now, let us focus on the following example of a set of rules that does not belong to Shy.

EXAMPLE 2.26. Consider the database $D = Q(a), P(a,b)$, and the following set Σ of rules.

$$\begin{array}{lrcl}
\sigma_1: & Q(x_1) & \rightarrow & \exists\, y_1\, U(x_1,y_1) \\
\sigma_2: & U(x_2,y_2), P(x_2,z_2) & \rightarrow & V(x_2,y_2,z_2) \\
\sigma_3: & V(x_3,y_3,z_3) & \rightarrow & P(x_3,y_3) \\
\sigma_4: & U(x_4,y_4) & \rightarrow & U(y_4,x_4)
\end{array}$$

As we have seen in Example 2.25, if the body of a rule contains only one atom, the conditions of Definition 2.24 are not violated. Hence, let us focus on rule σ_2. Computing the set of the invaded positions of Σ, we obtain that positions $U[1]$,$U[2]$, $V[2]$ and $P[2]$ are invaded by y_1. Accordingly, variables y_2 and z_2 are both attacked by y_1, and hence condition (2) of Definition 2.24 is not satisfied.

In plain words, we can say that the key idea behind this class is that during the execution of the chase over a shy set of rules and a database, nulls propagated body-to-head do not meet each other to join. Moreover, a null is propagated during a given fire, only from a single atom. In fact, condition (1) of Definition 2.24 guarantees that each variable occurring in more than one body atom is always mapped into a constant.

We refer the reader to Section 2.5 for the computational properties of this class.

2.3. The Notion of Proof Tree

In this section we describe the notion of proof tree that leads to space-bounded algorithms. This is a well known resolution-based method, that we use to show some fundamental results of this thesis. All the definition and properties that we introduce in the following can be found in the work of Berger et al. [23]. We point out that the literature about this topic is quite extensive, and for this reason, we refer the reader also to the conference papers [35, 37, 39, 51, 52, 59].

It is a popular fact that with a CQ q and a set Σ of TGDs, q can be unfolded by using the TGDs of Σ into an infinite union of CQs q_Σ in such a way that, for each database D, $\mathsf{cert}(q, D, \Sigma) = q_\Sigma(D)$.

The intent underlying the proof tree concept is to encode the sequence of CQs in a tree. This sequence of CQs which is generated during the unfolding of q with Σ, leads to a certain CQ q' of q_Σ such that every intermediate CQ, as well as q', is decomposed cautiously into smaller subqueries which form the tree nodes, while the root corresponds to q and leaves to q'.
It is interesting to observe that if we focus on well-behaved classes of TGDs such as *warded* (see Section 2.5) sets of TGDs, we can find upper bounds on the size of those subqueries, and, consequently, develop space-bounded algorithms for query answering.

The section is organized as follows. In Subsection 2.3.1 we define the notion of proof tree, and the main building blocks of it: chunk-based resolution, a query decomposition technique, and the notion of specialization for CQs, according to [23]. In Subsection 2.3.2 we recall the correspondence between proof trees and query answering.

2.3.1. Formal Definition. The goal of this subsection is to give the formal definition of proof tree. The first step needed to provide such definition, is to define a *chunk-based resolution* [23].

Intuitively, let us consider A and B to be two non-empty sets of atoms that mention only constants and variables. If there exists a substitution γ, which is the identity on Δ_C and such that $\gamma(A) = \gamma(B)$, we say that γ is a *unifier for A and B* and the sets A and B *unify*. A *most general unifier* which will be denoted with MGU, for A and B is a unifier for A and B, denoted $\gamma_{A,B}$, in such a way that for every unifier γ for A and B, $\gamma = \gamma' \circ \gamma_{A,B}$ for some substitution γ' [23]. The existence of a MGU, which is unique up to variable renaming, is guaranteed if two sets of atoms unify. Given a CQ $q(\mathbf{x})$ and a set of atoms $S \subseteq \mathsf{atoms}(q)$, we say that a variable $y \in \mathsf{vars}(S)$ is *shared*, if $y \in \mathbf{x}$ or $y \in \mathsf{vars}(\mathsf{atoms}(q) \setminus S)$. Consider a TGD σ that does not share a variable with q and two sets S_1 and S_2 such that $\emptyset \subset S_1 \subseteq \mathsf{atoms}(q)$, and $\emptyset \subset S_2 \subseteq \mathsf{head}(\sigma)$. Let γ be a unifier for S_1 and S_2 such that, for each $x \in \mathsf{vars}(S_2) \cap \mathsf{var}_\exists(\sigma)$,

(1) $\gamma(x) \notin \Delta_C$, i.e., $\gamma(x)$ is not constant, and
(2) for every variable y different from x, $\gamma(x) = \gamma(y)$ implies y occurs in S_1 and is not shared.

We say that the triple (S_1, S_2, γ) is a *chunk unifier* of q with a TGD σ. Moreover, if γ is MGU for S_1 and S_2, the triple (S_1, S_2, γ) is a *most general chunk unifier* (MGCU).

Consider the following example for clarifying the notion above.

EXAMPLE 2.27 ([23]). Consider the conjunctive query

$$q: \quad Q(x) \quad \leftarrow \quad R(x,y), S(y)$$

and the TGD

$$\sigma: \quad P(x') \quad \rightarrow \quad \exists\, y'\ R(x',y').$$

Resolving the atom $R(x,y)$ in q with σ via $\gamma = \{x \mapsto x', y \mapsto y'\}$ would be an unsound step due to the fact that the shared variable y is lost. This is because y' is unified with the shared variable y. On the other hand, $R(x,y), S(y)$ can be resolved with the TGD $\sigma' = P(x') \rightarrow \exists y' R(x',y'), S(y')$ using γ; hence, the chunk unifier is

$$(\mathsf{atoms}(q), \mathsf{head}(\sigma'), \gamma).$$

Now, we formally define a chunk-based resolution.

DEFINITION 2.28 (see Definition 4.3 in [23]). Let $q(\mathbf{x})$ be a CQ and σ a TGD. A σ-**resolvent** of q is a CQ $q'(\gamma(\mathbf{x}))$ with $\mathsf{body}(q') = \gamma((\mathsf{atoms}(q) \setminus S_1) \cup \mathsf{body}(\sigma))$ for a MGCU (S_1, S_2, γ) of q with σ.

As mentioned above, it is a known fact that, given a CQ q and a set Σ of TGDs, q can be unfolded by using the TGDs of Σ into an infinite union of CQs q_Σ such that, for each database D, $\mathsf{cert}(q, D, \Sigma) = q_\Sigma(D)$. The aim of a proof tree is the encoding a finite branch of the unfolding of a CQ q with a set Σ of TGDs, which is gotten through the application of chunk-based resolution. Such a branch is a sequence $q_0, \ldots, q_n$ of CQs, where $q = q_0$, while, for each $i \in [n]$, q_i is a σ-resolvent of q_{i-1} for some $\sigma \in \Sigma$.

Now, we give a simple example that illustrates the notion of unfolding.

EXAMPLE 2.29. (see Example 4.4 in [23]) Consider the following set $\Sigma = \sigma_1, \sigma_2, \sigma_3$ of TGDs, that will be used as running example in this section:

$$\begin{array}{rrcl}
\sigma_1: & R(x) & \rightarrow & \exists\, y\, T(x,y) \\
\sigma_2: & T(x,y), S(y,z) & \rightarrow & T(x,z) \\
\sigma_3: & T(x,y), P(y) & \rightarrow & G(),
\end{array}$$

and the CQ that simply asks whether $G()$ is entailed, i.e., the CQ

$$Q \leftarrow G().$$

It is noteworthy that unfolding q with Σ should provide the right answer for each input database. Let us consider the following database. For some $n > 1$:

$$\{R(c^{n-1}),\ S(c^{n-1}, c^{n-2}),\ \ldots\ ,\ S(c^2, c^1),\ P(c^1)\}.$$

One branch of the unfolding of q should be $q = q_0, \ldots, q_n$, where

$$\begin{array}{rcl}
q_0 = Q & \leftarrow & \underbrace{G()} \\
 & & \quad\downarrow \text{resolv } q_0 \text{ using } \sigma_3 \\
q_1 = Q & \leftarrow & \underbrace{T(x, y^1)}, P(y^1) \\
 & & \quad\downarrow \text{resolv } q_1 \text{ using } \sigma_2 \\
q_2 = Q & \leftarrow & \underbrace{T(x, y^2)}, S(y^2, y^1), P(y^1) \\
 & & \quad\downarrow \text{resolv } q_2 \text{ using } \sigma_2 \\
q_3 = Q & \leftarrow & \underbrace{T(x, y^3)}, S(y^3, y^2), S(y^2, y^1), P(y^1) \\
 & & \quad\vdots \text{ (resolv } q_3, \ldots, q_{n-2} \text{ using } \sigma_2) \\
q_{n-1} = Q & \leftarrow & \underbrace{T(x, y^{n-1})}, S(y^{n-1}, y^{n-2}), \ldots, S(y^2, y^1), P(y^1) \\
 & & \quad\downarrow \text{resolv } q_{n-1} \text{ using } \sigma_1 \\
q_n = Q & \leftarrow & R(y^{n-1}), S(y^{n-1}, y^{n-2}), \ldots, S(y^2, y^1), P(y^1).
\end{array}$$

It would seem that the proof tree is the finite labeled path $v_0, \ldots, v_n$, where each v_i is labeled by q_i. However, it is clear from the example above that by following the naive path encoding, we may get CQs of unbounded size [23]. The next objective of such a proof tree is to divide each resolvent q_i, for $i > 0$, into smaller subqueries $q_i^1, \dots, q_i^{n_i}$, in order to be treated independently by resolution. However, the problem in the process of the decomposition of q in subqueries is that, after its splitting, we may loose some join among variables, since they can be disjoined in different subqueries. The answer which was proposed by Berger et al. in [23] is only to distinguish occurrences of an output variable since this variable conventionally corresponds to a fixed constant value of Δ_C, and consequently it is never renamed by subsequent resolution steps. This ensure that it is possible to split occurrences of an output variable in different branch of the proof tree (without losing the semantic connection between them), while keeping together all the occurrences of a non-output variable, in order to preserve all the joins. Formally, a query decomposition is defined below.

Definition 2.30 (see Definition 4.5 in [23]). Given a CQ $q(\mathbf{x})$, a **decomposition** of q is a set of CQs

$$\{q_1(\mathbf{y_1}), \dots, q_n(\mathbf{y_n})\},$$

where $n \geq 1$ and $\cup_{i\in[n]}\mathsf{atoms}(q_i) = \mathsf{atoms}(q)$, such that, for each $i \in [n]$:

(1) $\mathbf{y_i}$ is the restriction of $\mathbf{x}$ on the variables in q_i , and

(2) for every $\underline{a}, \underline{b} \in \mathsf{atoms}(q)$, if $\underline{a} \in \mathsf{atoms}(q_i)$ and $\mathsf{vars}(\underline{a}) \cap \mathsf{vars}(\underline{b}) \subsetneq \mathbf{x}$, then $\underline{b} \in \mathsf{atoms}(q_i)$.

At this point, as shown in Example 2.29, we deal with the problem of not being able to split the query. In fact, in some cases, there is no way to understand that a non-output variable coincides with a fixed constant value, and, hence, that its occurrences could be split during the decomposition [23]. Indeed, the size of the CQs $\{q_i\}_{i>0}$ grows inconsistently, while the query decomposition does not have any effect on them since they are Boolean queries, i.e., queries which do not have output variables, and therefore, they cannot be divided into smaller subqueries. For this reason, a *specialization* step has been added, between resolution and decomposition. Intuitively, in this step, the transformation of some non-output variables into output variables while maintaining their name or bearing the name of an already existent output variable is done. Formally, this step is defined as follows.

Definition 2.31 (see Definition 4.6 in [23]). Let $q(\mathbf{x})$ be a CQ with $\mathsf{atoms}(q) = \{\underline{a}_1, \ldots, \underline{a}_n\}$. A **specialization** of q is a CQ $Q(\mathbf{x}, \mathbf{y}) \leftarrow \rho_{\mathbf{z}}(\underline{a}_1, \ldots, \underline{a}_n)$, where $\mathbf{y}, \mathbf{z}$ are (possibly empty) disjoint tuples of non-output variables of q, and $\rho_{\mathbf{z}}$ is a substitution from $\mathbf{z}$ to $\mathbf{x} \cup \mathbf{y}$.

Example 2.32 ([23]). Consider the CQ q_1 from Example 2.29

$$Q \leftarrow T(x, y^1), P(y^1)$$

obtained by resolving $q = q_0$ from rule σ_3 of Example 2.29.

As we can see, all the occurrences of the variable y^1 should be kept together (since it is a non-output variable), and hence this query cannot be decomposed into smaller subqueries.

Consider the following specialization of q_1

$$Q(y^1) \leftarrow T(x, y^1), P(y^1),$$

which converts y^1 into an output variable. Accordingly, the query can be decomposed into

$$Q(y^1) \leftarrow T(x, y^1), \text{ and}$$

$$Q(y^1) \leftarrow P(y^1).$$

Before we formulate the notion of proof tree, we introduce the following notational aids:

- given a partition $\pi = \{S_1, \ldots, S_m\}$ of a set of variables, eq_π denotes the substitution that maps the variables of S_i to the same variable x_i, where x_i is a distinguished element of S_i;
- given a CQ q and a TGD σ, an IDO σ-resolvent of q is a σ-resolvent of q such that the underlying MGCU makes use of a substitution which is the identity on the output variables of q;
- given a TGD σ and a node v, σ_v indicates the TGD deriving from the renaming of each variable x in σ into x_v.

Now, we have all the element to define the notion of proof tree, a resolution-based method of independent interest, that we will use to prove some complexity results in the thesis.

Definition 2.33 (see Definition 4.7 in [23]). Let $q(\mathbf{x})$ be a CQ with $\mathsf{atoms}(q) = \{\underline{a}_1, \ldots, \underline{a}_n\}$, and Σ a set of TGDs. A **proof tree** of q w.r.t. Σ is a triple $\mathcal{P} = (T, \lambda, \pi)$, where $T = (V, E)$ is a finite rooted tree, λ a labeling function that assign a CQ to each node of T, and π a partition of $\mathbf{x}$, such that, for $v \in V$:

(1) if v is the root node of T, then $\lambda(v)$ is the CQ $Q(\mathsf{eq}_\pi(\mathbf{x})) \leftarrow \mathsf{eq}_\pi(\underline{a}_1, \ldots, \underline{a}_m)$;

(2) if v has only one child u, $\lambda(u)$ is an IDO σ_v-resolvent of $\lambda(v)$ for some $\sigma \in \Sigma$, or a specialization of $\lambda(v)$.

(3) if v has the children $u_1, \ldots, u_k$ for $k > 1$, then $\{\lambda(u_1), \ldots, \lambda(u_k)\}$ is a decomposition of $\lambda(v)$;

Assuming that $v_1, \ldots, v_m$ are the leaf nodes of T, the CQ induced by $\mathcal{P}$ is definend as

$$Q(\mathsf{eq}_\pi(\mathbf{x})) \leftarrow \underline{a}_1, \ldots, \underline{a}_l,$$

where $\{\underline{a}_1, \ldots, \underline{a}_l\} = \cup_{i \in [m]} \mathsf{atoms}(\lambda(v_i))$.

The aim of the partition π in Definition 2.33 is to show that some output variables match an equal constant value. This is the reason of why variables in the same set of π are unified via substitution eq_π; this unification step is crucial in order to use substitutions that are identity on the output variables. If we skip this preliminary unification step, important resolution steps may be lost rendering it incomplete for the purpose of query answering.

Now, for completeness, we give an example that illustrates the notion of proof tree just introduced.

Example 2.34 (see Figure 1 in [23]). Proof tree that encodes the branch $Q = q_0, \ldots, q_n$ of the unfolding of q with Σ from Example 2.29.

$$\begin{tikzcd}
 & & & Q \leftarrow G() \arrow[d, dash] & \\
 & & & Q \leftarrow T(x,y^1), P(y^1) \arrow[d, dash] & \\
 & & & Q(y^1) \leftarrow T(x,y^1), P(y^1) \arrow[dl, dash] \arrow[dr, dash] & \\
 & & Q(y^1) \leftarrow T(x,y^1) \arrow[d, dash] & & \boxed{Q(y^1) \leftarrow P(y^1)} \\
 & & \vdots \arrow[d, dash] & & \\
 & & Q(y^{i-1}) \leftarrow T(x,y^{i-1}) \arrow[d, dash] & & \\
 & & Q(y^{i-1}) \leftarrow T(x,y^i), S(y^i,y^{i-1}) \arrow[d, dash] & & \\
 & & Q(y^{i-1},y^i) \leftarrow T(x,y^i), S(y^i,y^{i-1}) \arrow[dl, dash] \arrow[dr, dash] & & \\
 & Q(y^{i-1}) \leftarrow T(x,y^i) \arrow[d, dash] & & \boxed{Q(y^{i-1},y^i) \leftarrow S(y^i,y^{i-1})} & \\
 & \vdots \arrow[d, dash] & & & \\
 & Q(y^{n-2}) \leftarrow T(x,y^{n-2}) \arrow[d, dash] & & & \\
 & Q(y^{n-2}) \leftarrow T(x,y^{n-1}), S(y^{n-1},y^{n-2}) \arrow[d, dash] & & & \\
 & Q(y^{n-2},y^{n-1}) \leftarrow T(x,y^{n-1}), S(y^{n-1},y^{n-2}) \arrow[dl, dash] \arrow[dr, dash] & & & \\
Q(y^{n-1}) \leftarrow T(x,y^{n-1}) \arrow[d, dash] & & \boxed{Q(y^{n-2},y^{n-1}) \leftarrow S(y^{n-1},y^{n-2})} & & \\
\boxed{Q(y^{n-1}) \leftarrow R(y^{n-1})} & & & &
\end{tikzcd}$$

Finally we give the following:

DEFINITION 2.35 ([23]). The *node-width* of a proof tree $\mathcal{P}$ is defined as follows:

$$\mathsf{nwd}(\mathcal{P}) := max_{v \in V}\{|\lambda(v)|\}.$$

2.3.2. Main Properties.

In this section we report the main results obtained in the work of Berger et al. in [23], by denoting, with abuse of notation, $\mathcal{P}$ for the CQ induced by a proof tree $\mathcal{P}$.

The first result useful for our purposes concerns the connection between proof trees and CQ answering. What we intend to indicate is that verifying if a tuple $\mathbf{c}$ is a certain answer reduces to deciding if there exists a proof tree $\mathcal{P}$ in such a way that $\mathbf{c}$ is an answer to the CQ induced by $\mathcal{P}$ over the given database.

THEOREM 2.36 (see Theorem 4.8 in [23]). *Consider a database D, a set Σ of TGDs, a CQ $q(\mathbf{x})$, and a tuple $\mathbf{c} \in \mathsf{dom}(D)^{|\mathbf{x}|}$. Then the following are equivalent:*

(1) $\mathbf{c} \in \mathsf{cert}(q, D, \Sigma)$.

(2) There is a proof tree $\mathcal{P}$ of q w.r.t. Σ such that $\mathbf{c} \in \mathcal{P}(D)$.

In general, checking for the existence of such as proof tree is an undecidable problem. For this reason we focus on particular sets of TGDs and we search for the existence of a "well-behaved" proof trees, which enjoy certain syntactic properties which allow to design a decision procedure. We underline that this searching is strictly related to the class of TGDs under consideration. For this reason we postpone the discussion to Section 2.5 and to Chapter 4.

Then, we recall the auxiliary notion of *chase tree*; it operates as an intermediate structure between proof trees and chase sequences, enabling the use of the chase as an underlying technical tool. It is important to note that the notion of node-width can be clearly described for chase trees.

In order to introduce the notion of chase tree, for the completeness of the argument, in the sequel we report other needed notions [23].

Before anything else, we recall the concept of *chase graph*, and then we introduce the notion of *unraveling of the chase graph*, and then lastly, we introduce the notions of *unfolding* and *decomposition* for sets of atoms in the unraveling of the chase graph.

DEFINITION 2.37 ([23]). Fix a chase sequence $\delta = (I_i)_{i \geq 0}$ for a database D under a set Σ of TGDs with $I_i\langle\sigma_i, h_i\rangle I_{i+1}$, i.e., I_{i+1} is obtained by applying the firing homomorphism (σ_i, h_i) to I_i. The **chase graph** for D and Σ (w.r.t. δ) is a directed edge-labeled graph $\mathcal{G}^{D,\Sigma} = (V, E, \lambda)$, with λ being the labeling function, where $V = \mathsf{chase}(D, \Sigma)$, and an edge $(\underline{a}, \underline{b})$ labeled with (σ_k, h_k) belongs to E iff $\underline{a} \in h_k(\mathsf{body}(\sigma_k))$ and $\underline{b} \in I_{k+1} \setminus I_k$, for some $k \geq 0$.

In plain words, $\underline{a}$ has an edge to $\underline{b}$ if $\underline{b}$ is derived using $\underline{a}$, and, in addition, $\underline{b}$ is new in the sense that it has not been derived before. Note the fact that $\mathcal{G}^{D,\Sigma}$ does not have directed cycles. It is obvious that $\mathcal{G}^{D,\Sigma}$ depends on δ, but a fixed sequence δ can be assumed since each chase sequence leads to the same outcome (up to isomorphism) as earlier mentioned in Section 1.4.

Now, we report the notion of unraveling of the chase graph. Given a set $\Theta \in V$ of nodes, the **unraveling of $\mathcal{G}^{D,\Sigma}$ around Θ** is, informally, the directed node- and edge-labeled forest $\mathcal{G}_{\Theta}^{D,\Sigma} = (V_{\Theta}, E_{\Theta}, \mu_{\Theta})$, that has a tree for each $\underline{a} \in \Theta$ whose branches are backward-paths in $\mathcal{G}^{D,\Sigma}$ from $\underline{a}$ to a database atom. The edges between nodes are labeled by pairs (σ, h) just like in $\mathcal{G}^{D,\Sigma}$, while the nodes are labeled by atoms and, importantly, the atoms along the paths in $\mathcal{G}^{D,\Sigma}$ may be duplicated and labeled nulls are given new names. We write $U(\mathcal{G}^{D,\Sigma}, \Theta)$ for the set of all atoms that appear as labels in $\mathcal{G}_{\Theta}^{D,\Sigma}$. [23]

DEFINITION 2.38 ([23]). Given a node $v \in \mathcal{G}_{\Theta}^{D,\Sigma}$, we denote by $\mathsf{succ}_{\sigma,h}(v)$ the set of labels of all children of v whose edges from v are labeled by (σ, h).

DEFINITION 2.39 ([23]). Let $\Gamma, \Gamma' \subseteq U(\mathcal{G}^{D,\Sigma}, \Theta)$. We say that Γ' is an **unfolding** of Γ if there are $\underline{a} \in \Gamma$ and $\underline{b}_1, \ldots, \underline{b}_k \in U(\mathcal{G}^{D,\Sigma}, \Theta)$ such that:

(1) $\mathsf{succ}_{\sigma,h}(v) = \{\underline{b}_1,\dots,\underline{b}_k\}$, for some $\sigma \in \Sigma$ and h, and some node $v \in \mathcal{G}_{\Theta}^{D,\Sigma}$ labeled with $\underline{a}$;

(2) for each null which occurs in $\underline{a}$, either it does not appear in $\Gamma \setminus \{\underline{a}\}$, or it appears in $\{\underline{b}_1,\dots,\underline{b}_k\}$;

(3) $\Gamma' = (\Gamma \setminus \underline{a}) \cup \{\underline{b}_1,\dots,\underline{b}_k\}$.

Definition 2.40 ([23]). Let $\Gamma \in U(\mathcal{G}^{D,\Sigma},\Theta)$ be a non-empty set. A **decomposition** of Γ is a set $\Gamma_1,\dots,\Gamma_n$, for $n \geq 1$, of non-empty subsets of Γ such that:

(1) $\Gamma = \cup_{i\in[n]}\Gamma_i$, and

(2) $i \neq j$ implies that Γ_i and Γ_j do not share a labeled null.

Now we are ready to formally define the key notion of chase tree.

Definition 2.41 (see Definition 4.11 in [23]). Let D be a database, Σ a set of TGDs, $\Theta \subseteq \mathsf{chase}(D,\Sigma)$ and $\Gamma \subseteq U(\mathcal{G}^{D,\Sigma},\Theta)$. A **chase tree** for Γ (w.r.t. $\mathcal{G}_{\Theta}^{D,\Sigma}$) is a pair $C = (T,\lambda)$, where $T = (V,E)$ is a finite rooted tree and λ is a labeling function that assigns a subset of $U(\mathcal{G}^{D,\Sigma},\Theta)$ to each node of T, such that, for each $v \in V$, the following hold:

(1) if v is the root node of T, then $\lambda(v) = \Gamma$;

(2) if v has only one child v', then $\lambda(v')$ is an unfolding of $\lambda(v)$;

(3) if v has more than one child $v'_1,\dots,v'_k$ for $k > 1$, then $\{\lambda(v'_1),\dots,\lambda(v'_k)\}$ is a decomposition of $\lambda(v)$;

(4) if v is the leaf node, then $\lambda(v) \subseteq D$;

The *node-width* C is defined as $\mathsf{nwd}(C) := max_{v\in V}\{|\lambda(v)|\}$.

Finally, we exploit the following theorem to connect the chase tree and the proof tree.

Theorem 2.42 (see Lemma 4.13 in [23]). *Consider a database D and a set Σ of TGDs. Let $\Theta \subseteq \mathsf{chase}(D,\Sigma)$, $q(\mathbf{x})$ be a CQ, and $\mathbf{c}$ be a tuple of constants such that $h(atoms(q)) \subseteq U(\mathcal{G}^{D,\Sigma},\Theta)$ and $h(\mathbf{x}) = \mathbf{c}$,* *for some homomorphism h. If there is a chase tree C for $h(atoms(q))$ with $\mathsf{nwd}(C) \leq m$, then there is a proof tree $\mathcal{P}$ for q w.r.t. Σ such that $\mathsf{nwd}(\mathcal{P}) \leq m$ and $\mathbf{c} \in \mathcal{P}(D)$.*

2.4. The Ward Class

In this section we introduce the class Ward, that is a member of the Datalog± family of knowledge representation languages [35] and forms a subclass of a highly expressive class of TGDs known in the literature as *weakly-frontier-guarded* sets of TGDs [16]. In particular, this rule-based formalism represents the logical core of the Vadalog system [21, 22, 53, 54]. This fragment captures plain Datalog as well as SPARQL queries under the entailment regime for OWL 2 QL [45], and it is able to perform ontological reasoning tasks. In fact, in the TGD literature, it is well known that an uncontrolled use of dangerous variables can lead to an high computational complexity of CQ answering [27]. Hence, we are interested to this class, since it aims at taming the way that nulls are propagated during the chase.

We observe that the syntactic condition of the class Ward is defined on the classical notion of affected position reported in the following.

Definition 2.43. ([23]) Consider a set Σ of TGDs. The set of **affected position** of $\mathsf{sch}(\Sigma)$, denoted by $\mathsf{aff}(\Sigma)$, is inductively defined as follows:

(1) if there exists $\sigma \in \Sigma$ and a variable $x \in \mathsf{var}_{\exists}(\sigma)$ at position π, then $\pi \in \mathsf{aff}(\Sigma)$, and

(2) if there exists $\sigma \in \Sigma$ and a variable $x \in \mathsf{front}(\sigma)$ in the body of σ only at positions of $\mathsf{aff}(\Sigma)$, and x appears in the head of σ at position π, then $\pi \in \mathsf{aff}(\Sigma)$.

Let $\mathsf{nonaff}(\Sigma) = \mathsf{pos}(\Sigma) \setminus \mathsf{aff}(\Sigma)$. We can give a classification of the variables that appear in the body of a TGDs.

Definition 2.44. ([23]) Consider a set Σ of TGDs. Fix $\sigma \in \Sigma$ and a variable x in $\mathsf{body}(\sigma)$:

- x is **harmless** if at least one occurrence of it appears in $\mathsf{body}(\sigma)$ at a position of $\mathsf{nonaff}(\Sigma)$;

- x is **harmful** if it is not harmless;
- x is **dangerous** if it is harmful and belongs to $\mathsf{front}(\sigma)$.

Now we can formally define the class as follows.

DEFINITION 2.45. (*Wardedness*, see Definition 3.1 in [23]) A set Σ of TGDs is **warded** if, for each $\sigma \in \Sigma$, there are no dangerous variables in $\mathsf{body}(\sigma)$, or there exists an atom $\underline{a} \in \mathsf{body}(\sigma)$, called a *ward*, such that:

(1) all the dangerous variables in $\mathsf{body}(\sigma)$ occur in $\underline{a}$, and

(2) each variable in $\mathsf{vars}(\underline{a}) \cap \mathsf{vars}(\mathsf{body}(\sigma) \setminus \{\underline{a}\})$ is harmless.

We write Ward for the class of all finite warded sets of TGDs.

EXAMPLE 2.46. Consider the following set Σ of rules:

$$\begin{array}{lrcl} \sigma_1: & P(x_1,y_1), S(y_1,z_1) & \rightarrow & \exists\, w_1\; T(y_1,x_1,w_1) \\ \sigma_2: & T(x_2,y_2,z_2) & \rightarrow & \exists\, w_2\; P(w_2,x_2) \\ \sigma_3: & T(x_3,y_3,z_3) & \rightarrow & S(x_3,y_3) \end{array}$$

We observe that in the first rule the variable y_1 is not dangerous, since it appears in positions $P[2]$ and $S[1]$, that are not affected; hence x_1 is dangerous, while y_1 and z_1 are harmless. It follows that, since variable x_1 appears only in one body atom, and the join is on an harmless variable, the wardedness conditions are satisfied. Rules σ_2 and σ_3 are trivially warded, since in the body is contained only one atom.

Differently from the class Shy, defined only for single-head TGDs, the class Ward allows multi-head TGDs. However, from now on, we will focus only on single-head TGDs, since we can always normalize a warded set of TGDs into single-head TGDs, preserving the certain answers. The normalization works as follow [23].

Consider a TGD σ of the form

$$\sigma: \; \Phi(\mathbf{x},\mathbf{y},\mathbf{z}) \rightarrow \exists\, \mathbf{w}\; P_1(\mathbf{x}_1,\mathbf{w}_1,\mathbf{z}_1), \ldots, P_n(\mathbf{x}_n,\mathbf{w}_n,\mathbf{z}_n)$$

where $\mathbf{x}_i \subseteq \mathbf{x}$, $\mathbf{z}_i \subseteq \mathbf{z}$ and $\mathbf{w}_i \subseteq \mathbf{w}$, for each $i \in \{1, \ldots, n\}$.

Let $\mathsf{SH}(\sigma)$ the set of single-head TGDs consisting of

$$\begin{array}{rcl} \Phi(\mathbf{x},\mathbf{y},\mathbf{z}) & \rightarrow & \exists\ \mathbf{w}\ Aux_\sigma(\mathbf{x},\mathbf{z},\mathbf{w}) \\ Aux_\sigma(\mathbf{x},\mathbf{z},\mathbf{w}) & \rightarrow & P_1(\mathbf{x}_1,\mathbf{w}_1,\mathbf{z}_1) \\ & \vdots & \\ Aux_\sigma(\mathbf{x},\mathbf{z},\mathbf{w}) & \rightarrow & P_n(\mathbf{x}_1,\mathbf{w}_1,\mathbf{z}_1). \end{array}$$

The normalization of Σ into single-head TGDs, is defined as

$$\mathcal{N}_{sh}(\Sigma) = \bigcup_{\sigma \in \Sigma} \mathsf{SH}(\sigma).$$

2.5. Shy vs. Ward

In this section we analyse the key properties and differences between the classes Shy and Ward, previously presented, since they will be our starting point for the definition of a new fragment, for which we refer the reader to Chapter 4. Both languages extend Datalog by existential quantifiers in rule heads, but restricts, at the same time, their syntax in order to achieve decidability and data tractability (see e.g. [6, 27, 35, 29, 33]). In this thesis, we choose to focus on these two classes since both of them offer a good balance between expressivity and complexity, and are suitable for an efficient implementation.

The first difference that is possible to note, is that the two classes are incomparable. In fact, it is not possible that one of the two contains the other, since condition (1) of Definition 2.45, that is, all the dangerous variables occur in the ward, goes againts condition (2) of Definition 2.24, that allows the occurrence of dangerous variables in different atoms. Therefore, unless we consider trivial cases, one condition excludes the other. However, the intersection between Shy and Ward is not empty, since both generalize Datalog as well as the class of linear TGDs (i.e., with at most one body-atom). Hence, intuitively, the situation is well described by Figure 1.

In order to explain better the relation between the two classes, as argued above, we provide some examples:

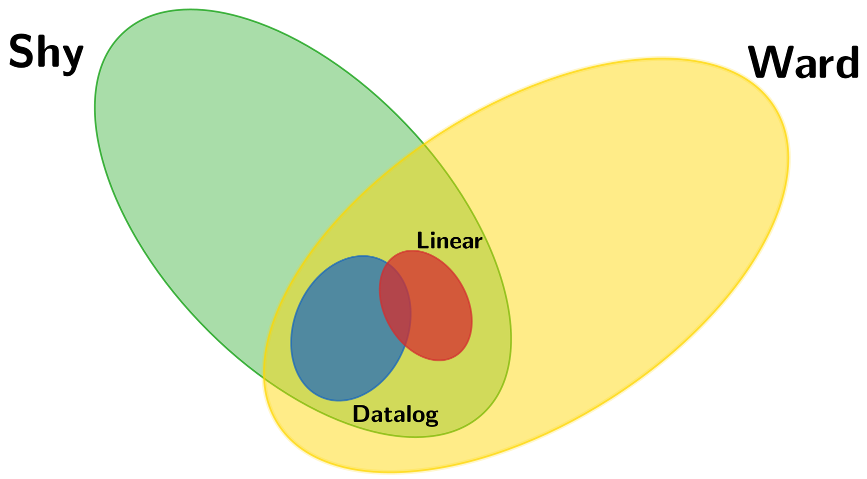


FIGURE 1. Syntactic relation between Shy and Ward.

EXAMPLE 2.47. The following set of TGDs belongs to Ward, but not to Shy.

$$\begin{array}{lrcl}
\sigma_1: & S(x_1) & \to & \exists\, y_1\ T(x_1,y_1)\\
\sigma_2: & T(x_2,y_2),Q(z_2) & \to & \exists\, w_2\ P(y_2,w_2,z_2)\\
\sigma_3: & P(x_3,y_3,z_3),S(z_3),V(u_3,w_3),K(w_3) & \to & R(x_3,u_3,y_3)\\
\sigma_4: & U(x_4) & \to & \exists\, y_4\ V(x_4,y_4)\\
\sigma_5: & V(x_5,y_5) & \to & K(y_5)
\end{array}$$

We observe that rules $\sigma_1, \sigma_2, \sigma_5$ are linear, hence they are both shy and warded rules. In rule σ_2 the dangerous variable y_2 is contained in a single ward and the same hols for rule σ_3. However, rule σ_3 violates condition (1) of Definition 2.24 since variable w_3, that occurs in two body atoms, is not protected.

EXAMPLE 2.48. The following set of TGDs belongs to Shy, but not to Ward.

$$\begin{array}{lrcl}
\sigma_1: & P(x_1) & \to & \exists\, y_1\ T(y_1)\\
\sigma_2: & Q(x_2) & \to & \exists\, y_2\ S(x_2,y_2)\\
\sigma_3: & T(x_3),S(y_3,z_3) & \to & R(x_3,y_3,z_3)
\end{array}$$

We observe that rules σ_1 and σ_2 are linear, hence they are both shy and warded rules, but rule σ_3 violates the wardedness condition, since there is no a ward atom that contains both the dangerous variables x_3

and z_3. On the other hand, rule σ_3 satisfies shyness condition, since the variable x_3 and z_3 are, respectively, attacked by y_1 and y_2.

EXAMPLE 2.49. The following set of TGDs belongs to Shy and Ward.

$$\begin{array}{lrcl}\sigma_1: & P(x_1,y_1),S(y_1,z_1) & \rightarrow & \exists\, w_1\; T(y_1,x_1,w_1)\\ \sigma_2: & T(x_2,y_2,z_2) & \rightarrow & \exists\, w_2\; P(w_2,z_2)\\ \sigma_3: & T(x_3,y_3,z_3) & \rightarrow & S(x_3,y_3)\end{array}$$

We note that σ_2 and σ_3 are linear, hence they are both shy and warded rules, while rule σ_1 satisfies both shyness and wardedness conditions.

More formally, we can deduce that it holds the following result:

PROPOSITION 2.50. *The classes* Shy *and* Ward *are syntactically incomparable.*

Below, we summarize[1] all the strengths of Shy and Ward, each of which convinced us to consider these classes as a starting point for the definition of the new fragment that we will present in Chapter 4:

- they offer good expressiveness strictly generalizing Datalog;
- they are QA-decidable;
- they are efficiently computable;
- they aim at taming the way that nulls are propagated during the chase;
- they are suitable for an efficient implementation.

Now, we conclude this section by reporting the complexity results regarding Shy and Ward, obtained, respectively, in [63, 64] and [11, 23, 53].

THEOREM 2.51 (see Theorem 5.5 in [63]). *Checking whether a set of TGDs over a database D is* Shy *is decidable. In particular, this check is doable in polynomial time.*

[1]For more detail we refer the reader to the conference papers [23] and [63].

THEOREM 2.52 ([64]). *Problem* CQAns(Shy) *is decidable. In particular,*

(1) it is PTIME*–complete in data complexity, and* EXPTIME*-complete in combined complexity;*

(2) given a database D, a shy set of TGDs and a BCQ q, it holds that

$$D \cup \Sigma \models q \Leftrightarrow \mathsf{pchase}(D \cup \Sigma, |\mathsf{vars}(q)|+1) \models q.$$

REMARK 2.53. It has been shown in [63] that to compute $\mathsf{pchase}(D \cup \Sigma, n+1)$, the number of steps performed by the parsimonious chase after n resumptions is at most:

$$(n+1)(c+\alpha)^{\alpha^{2(n+1)}},$$

where $c = \mathsf{const}(D)$, $\beta = \max_{\sigma \in \Sigma} |\mathsf{body}(\sigma)|$, $w = \mathsf{arity}(\Sigma)$, $\alpha = \max\{|\Sigma|, |\mathsf{sch}(\Sigma)|, w+1, \beta+2\}$, and $n = |\mathsf{vars}(q)|$.

Clearly, in data complexity, both n and α are considered fixed. Then, the number of steps to execute n resumptions is polynomial in c, while it is double exponential in combined complexity.

We want to stress the statement (2) of the above theorem. To better understand the idea behind the statement, we propose the following example.

EXAMPLE 2.54. Consider the following set Σ of rules:

$$\begin{array}{lrcl}
\sigma_1 : & U(x_1,y_1) & \rightarrow & \exists\, z_1\; V(z_2) \\
\sigma_2 : & V(x_2) & \rightarrow & \exists\, y_2\; U(x_2,y_2) \\
\sigma_3 : & V(x_3), P(y_3,z_3) & \rightarrow & P(x_3,z_3) \\
\sigma_4 : & P(x_3,y_3), U(z_3,w_3) & \rightarrow & P(x_3,w_3)
\end{array}$$

Consider the BCQ $q = \exists\, x,y\; P(x,y), U(x,y)$.

By computing only $\mathsf{pchase}(D \cup \Sigma)$, we are not able to find an answer to q, since we obtain the set:

$$\mathsf{pchase}(D \cup \Sigma) = \{V(\varphi_1), P(a,d)\}.$$

However, by iteratively resuming the pchase a number of times that depends on the number of distinct variables in the query, it is possible to deal with joins in the query, and find an answer for q. In particular, we can see that this ontology Σ over D requires the computation of $\mathsf{pchase}(D \cup \Sigma, 3)$ to prove (after two resumptions) that a q containing two atoms and two variables is true over $D \cup \Sigma$.

$$\begin{aligned}
&\mathsf{pchase}(D \cup \Sigma, 0) = D = \{P(a,b), U(c,d)\}\\
&\mathsf{pchase}(D \cup \Sigma, 1) = \mathsf{pchase}(D \cup \Sigma) = \{V(\varphi_1), P(a,d)\}\\
&\mathsf{pchase}(D \cup \Sigma, 2) = \{U(\varphi_1, \varphi_2), P(\varphi_1, b)\}\\
&\mathsf{pchase}(D \cup \Sigma, 3) = \{P(\varphi_1, \varphi_2)\}
\end{aligned}$$

The next result, regarding the complexity of the class , has been explored in the conference papers [11, 53] for the data complexity. Nevertheless, an ExpTime upper bound in combined complexity can be deduced by the same algorithm, while the lower bounds are inherited from Datalog since a set of Datalog rules (seen as TGDs) is warded. Hence, we have the following:

Theorem 2.55 (see Proposition 3.2 in [23]). $\mathsf{CQAns}(\mathsf{Ward})$ *is* ExpTime-*complete in combined complexity, and* PTime-*complete in data complexity.*

More recently in [23], via the use of the proof tree (see Section 2.3), an alternative way has been provided to establish the complexity of the class Ward, stated in the following:

Theorem 2.56 (see Theorem 4.8 in [23]). *Consider a database D, a set Σ of warded TGDs, a CQ $q(\mathbf{x})$, and a tuple $\mathbf{c} \in \mathsf{dom}(D)^{|\mathbf{x}|}$. Let $\Sigma' = \mathcal{N}_{sh}(\Sigma)$. Then the following are equivalent:*

(1) $\mathbf{c} \in cert(q, D, \Sigma)$.

(2) There exists a proof tree $\mathcal{P}$ of q w.r.t. Σ' with $\mathsf{ndw}(\mathcal{P}) \leq f_{WARD}(q, \Sigma')$[2] *such that $\mathbf{c} \in \mathcal{P}(D)$.*

[2] $f_{\mathrm{WARD}}(q, \Sigma') := 2 \cdot \max\{|q|, \max_{\sigma \in \Sigma'}\{|\mathsf{body}(\sigma)|\}\}$, for more details see [23].

Thanks to this result, given a database D, a set $\Sigma \in \mathsf{Ward}$, a CQ $q(\mathbf{x})$ and a tuple $\mathbf{c} \in \mathsf{dom}(D)^{|\mathbf{x}|}$, the problem reduces to check if there exists a proof tree $\mathcal{P}$ of q w.r.t. Σ with $\mathsf{ndw}(\mathcal{P}) \leq f_{\text{WARD}}(q, \Sigma')$. The last can be checked via a space-bounded algorithm, proposed in [24].

ALGORITHM 2 ([24]). Alternating algorithm for $\mathsf{CQAns}(\mathsf{WARD})$

INPUT: A database D, a set of TGDs $\Sigma \in \mathsf{Ward}$, a CQ $q(\mathbf{x})$, and a tuple $\mathbf{c} \in \text{dom}(D)^{|\mathbf{x}|}$
OUTPUT: Accept if $\mathbf{c} \in \text{cert}(q, D, \Sigma)$; Otherwise, Reject

```
 (1) $\Sigma := \mathcal{N}_{sh}(\Sigma)$;
 (2) $p := Q \leftarrow \underline{a}_1, \cdots, \underline{a}_n$, with $\mathsf{atoms}(q(\mathbf{c})) = \{\underline{a}_1, \cdots, \underline{a}_n\}$ ;
 (3) repeat
 (4)     if $\mathsf{atoms}(p) \subseteq D$ then
 (5)       Accept
 (6)     guess $op \in \{r, d, s\}$
 (7)     if $op = r$ then
 (8)       guess a TGD $\sigma \in \Sigma$
 (9)       if $\text{mgcu}(p, \sigma) = \emptyset$ then
(10)         Reject
(11)       else
(12)           guess $U \in \text{mgcu}(p, \sigma)$
(13)           if $|p[\sigma, U]| > f_{\text{WARD}}$ then
(14)             Reject
(15)            else
(16)               $P := \{p[\sigma, U]\}$
(17)     if $op = d$ then
(18)       guess a decomposition $P$ of $p$
(19)     if $op = s$ then
(20)       guess $V \subseteq \mathsf{vars}(p)$ and $\gamma : V \to \mathsf{dom}(D)$
(21)       $P := \{\gamma(p)\}$
(22)     universally select every CQ $p \in P$
(23) until False;
```

This alternating algorithm uses polynomial space in general and logarithmic space in data complexity. Hence, we get the EXPTIME (resp., PTIME) upper bound in combined (resp., data) complexity.

We can summarize the previous results in the following table.

	Data Complexity	Combined Complexity
Shy	PTime-c	ExpTime-c
Ward	PTime-c	ExpTime-c

For any complexity class $\mathcal{C}$, for brevity,
$\mathcal{C}$-c is used as a shorthand for $\mathcal{C}$-complete.

2.6. Other Existing Decidable Classes

As discussed in the previous chapters of the thesis, in the literature there are different extensions of Datalog. In particular, in 2009 a family of Datalog-based languages for tractable query answering over ontologies has been proposed, named Datalog$^{\pm}$. The aim of this family is to collect all expressive extensions of Datalog which are based on TGDs, equality-generating dependencies and negative constraint. In particular, the “plus” symbol refers to any possible combination of these extensions, while the “minus” one imposes at least decidability. In what follows, we give a survey of these existent classes.

Syntactic and Semantic Conditions

In order to classify the several decidable classes in the literature, semantic classes have been defined, based on the behaviour of reasoning mechanisms [15]. These classes do not come with a syntactic property that can be checked on rules and, moreover, they are not recognizable, i.e., the problem of determining if a given set of rules satisfies the abstract properties is not decidable [14].

We can summarize these classes as follows:

- *Finite Expansions Sets (FES)*: sets of TGDs which ensure the termination of the chase;
- *Bounded Treewidth Sets (BTS)*: sets of TGDs which guarantee that the (possibly infinite) instance constructed by the chase has bounded treewidth;

- *Finite Unification Sets (FUS)*: sets of TGDs which guarantee the termination of (resolution-based) backward chaining procedures;

- Parsimonious Sets (PS): sets of TGDs under which the chase can be terminated early.

More precisely, we stress that the notion of FUS is strictly related to that of rewriting. Given a set of TGDs Σ and a BCQ q, a backward chaining mechanism is a procedure that constructs a rewriting q_Σ of q relative to Σ, also called Σ-rewriting of q, such that for every database D, $D \cup \Sigma \models q \Leftrightarrow D \models q_\Sigma$. The key operation in backward chaining is the unification between the set of atoms in the body of q and the head of some TGD in Σ (see the works of Baget et al. [16] and Mugnier [71] for more details). Regarding the notion of BTS we remark that a set Σ is BTS iff for every database D, the chase graph of $\mathsf{chase}(D, \Sigma)$ has bounded treewidth, i.e., the chase graph is a "tree-like" graph. Finally, a set of TGDs belongs to FES iff for each set D of ground facts, $D \cup \Sigma$ admits a finite universal model.

We point out that each of the above conditions has also its syntactic counterpart: just to name a few, we recall that *linear* and *sticky* rules are FUS [31], *weakly-acyclic* rules are FES [44], every set of linear, *guarded*, and *weakly guarded* TGDs is a BTS [27], while shy rules are PS [63].

Syntactic Relationship Among Classes

In this paragraph we survey the syntactic subclasses of the abstract classes defined above. We first recall some concrete subclasses of BTS. The first syntactic subclass that we mention is linear, according to which at most one body atom is allowed in each rule; this class generalise the known class inclusion dependencies (see the works of Abiteboul et al. [1] and Johnson et al. [58] for more details); the second one is guarded, according to which each rule needs at least one body atom that covers all universal variables; the third one is weakly-guarded, that extends both guarded and datalog by allowing unaffected "unguarded" variables. Then, the class frontier-guarded has been defined, that is a generalization of guarded, since the syntactic condition has been weakened by requiring that each rule needs at least one body atom that covers all frontier variables; furthermore, weakly-frontier-guarded [14] generalizes both frontier-guarded and weakly-guarded by requiring that each rule needs at least one body atom that covers all frontier variables classified as affected.

Among the concrete subclasses of FUS we recall the well-known class sticky, defined by Calì et al. [31]. It enjoys very good complexity, but it does not capture datalog. Intuitively, if a program is sticky, then all of the atoms that are inferred (by the chase) starting from a given join contain the term of this join. Subsequently, the class sticky-join [32] have been refined by the same authors, who manage to preserve the same complexity of sticky, while generalizing linear. Conversely, Gogacz and Marcinkowski[46] introduced joinless, the subclass of sticky collecting all and only the programs where each body contains no repeated variables. Then, we mention also the following classes: multi-linear [29], where, in each rule, every body atom contains all universal variables, and weakly-recursive [40] relying on a novel notion of *position graph* and generalizing all mentioned concrete FUS classes.

Among concrete subclasses of FES, we recall weakly-acyclic that has been introduced by Fagin et al. [44]. Roughly speaking, a program is weakly acyclic if the presence of a null occurring in an inferred atom at a given position does not trigger the inference of an infinite number of atoms (with the same relational predicate) containing several nulls in the same position. A generalization of this class has been defined with the class jointly-acyclic [61]. A number of extensions, techniques, and further criteria for checking chase termination have also been proposed in this context [41, 55, 56, 69, 70]. For know more about other subclasses of FES, we refer the reader to [55] which reviews and compares many previous results on acyclicity notions and introduces model-faithful-acyclic, the most general concrete FES class currently known. Moreover, we refer the reader also to the work of Leone et al. [64], for a formal proof regarding the taxonomy above described.

We now summarize the computational complexity of the aforementioned classes.

Main syntactic classes	Data Complexity	Combined Complexity
weakly (fr) guarded	ExpTime-c	2ExpTime-c
(fr) guarded	PTime-c	2ExpTime-c
weakly-acyclic	PTime-c	2ExpTime-c
jointly-acyclic	PTime-c	2ExpTime-c
datalog	PTime-c	ExpTime-c
shy	PTime-c	ExpTime-c
ward	PTime-c	ExpTime-c
ward$^+$	PTime-c	ExpTime-c
sticky	in AC_0	ExpTime-c
sticky-join	in AC_0	ExpTime-c
linear	in AC_0	PSpace-c
joinless	in AC_0	PSpace-c
inclusion-dependencies	in AC_0	PSpace-c

For any complexity class $\mathcal{C}$, for brevity, $\mathcal{C}$-c is used as a shorthand for $\mathcal{C}$-complete.

CHAPTER 3

Dyadic TGDs

In this chapter we present the main theoretical contribution of this thesis, that is the definition of a new decidable paradigm for ontological query answering, called *Dyadic TGDs*. Our claim is to combine and generalize existing decidable classes, in order to exploit the systems already developed. In particular, given a decidable class $\mathcal{C}$, we define a generalization of it, via the class Dyadic-$\mathcal{C}$.

The chapter is structured as follows. In Section 3.1 we present the class Dyadic-$\mathcal{C}$ and we define what is a *dyadic decomposition* $(\Sigma_{\mathrm{HG}}, \Sigma_{\mathcal{C}})$ for a set Σ of TGDs w.r.t some decidable class $\mathcal{C}$. In Section 3.2 we prove that the class Dyadic-$\mathcal{C}$ is decidable, providing a sound and complete algorithm used to complete the database with all the ground atoms that are possible to derive from the component Σ_{HG} of the dyadic decomposition, in order to exploit only the component $\Sigma_{\mathcal{C}}$ for query answering purposes.

3.1. Formal Definition

In the same spirit of the invaded position recalled in Definition 2.22, we generalize the notion of affected position mentioned in Definition 2.43, proposed to separate the positions in which the chase can introduce only constants from those where nulls might appear. We remember that we assume that the same variable does not occur in two different rules of Σ.

For the reader convenience, we now recall the classical definition of affected position.

Definition 2.43 ([23]). Consider a set Σ of TGDs. The set of **affected position** of $\mathsf{sch}(\Sigma)$, denoted by $\mathsf{aff}(\Sigma)$, is inductively defined as follows:

(1) if there exists $\sigma \in \Sigma$ and a variable $x \in \mathsf{var}_\exists(\sigma)$ at position π, then $\pi \in \mathsf{aff}(\Sigma)$, and

(2) if there exists $\sigma \in \Sigma$ and a variable $x \in \mathsf{front}(\sigma)$ in the body of σ only at positions of $\mathsf{aff}(\Sigma)$, and x appears in the head of σ at position π, then $\pi \in \mathsf{aff}(\Sigma)$.

Having in mind this notion, we give the following generalization.

Definition 3.1. Consider a set Σ of TGDs and a variable $z \in \mathsf{var}_\exists(\Sigma)$. A position $R[i]$ is z-affected if one between these two properties hold:

(1) there exists $\sigma \in \Sigma$ such that z appears in the head of σ at position $R[i]$;

(2) there exist $\sigma \in \Sigma$ and $x \in \mathsf{front}(\sigma)$ s.t. x occurs both in $\mathsf{head}(\sigma)$ at position $R[i]$ and in $\mathsf{body}(\sigma)$ at z-affected positions only.

A position π is **S-affected**, where $S \subseteq \mathsf{var}_\exists(\Sigma)$, if:

(1) for each $x \in S$, π is x-affected, and

(2) for each $x \in \mathsf{var}_\exists(\Sigma)$, if π is x-affected, then $x \in S$.

Remark 3.2. For every position π there exists a unique set S such that π is S-affected. We write $\mathsf{aff}(\pi)$ for this set S. Moreover,

$$\mathsf{aff}(\Sigma) = \{\pi \in \mathsf{pos}(\Sigma) \mid \mathsf{aff}(\pi) \neq \emptyset\},$$

and

$$\mathsf{nonaff}(\Sigma) = \mathsf{pos}(\Sigma) \setminus \mathsf{aff}(\Sigma).$$

The following example compares the notion of S-affected position (see Definition 3.1) with the classical one (see Definition 2.43).

Example 3.3. Consider a set Σ of TGDs consisting of the following rules.

$$\begin{array}{lrcl}
\sigma_1 : & & \rightarrow & \exists\, x_1\ P(x_1) \\
\sigma_2 : & & \rightarrow & \exists\, x_2\ T(x_2) \\
\sigma_3 : & P(x_3), T(x_3) & \rightarrow & R(x_3) \\
\sigma_4 : & R(x_4), R(y_4) & \rightarrow & S(x_4, y_4)
\end{array}$$

The positions $P[1]$ and $T[1]$ are, respectively, x_1-affected and x_2-affected. According to Definition 3.1, we have that the position $R[1]$ is not affected by any variable, since

$$\mathsf{aff}(P[1]) \cap \mathsf{aff}(T[1]) = \emptyset.$$

Hence, there does not exist $x \in \mathsf{var}_\exists(\Sigma)$ such that $R[1]$ is x-affected, and

$$\mathsf{aff}(\Sigma) = \{P[1], T[1]\}.$$

Conversely, according to Definition 2.43, we would obtain that

$$\mathsf{aff}(\Sigma) = \{P[1], T[1], T[1], S[1], S[2]\}.$$

Therefore, with this generalization we are able to catch the "really" affected positions, i.e., positions containing variables that could be really mapped to some null.

We can now categorize the variables in the body of a TGDs.

Definition 3.4. Given a TGD $\sigma \in \Sigma$ and a variable x in $\mathsf{body}(\sigma)$:

- if x occurs at positions $\pi_1, \ldots, \pi_n$ and $\bigcap_{i=1}^n \mathsf{aff}(\pi_i) = \emptyset$, then x is **harmless**,
- if x is not harmless, placed $S = \bigcap_{i=1}^n \mathsf{aff}(\pi_i)$, then it is S-**harmful**,
- if x is S-harmful and belongs to $\mathsf{front}(\sigma)$, then x is S-**dangerous**.

Every variable x that is S-dangerous (resp. S-harmful), for some $S \neq \emptyset$, is also dangerous (harmful). □

We notice that, given a variable x that is S-dangerous, we write $\mathsf{dang}(x)$ for the set S. Given a rule $\sigma \in \Sigma$, we write $\mathsf{dang}(\sigma)$ ($\mathsf{harmless}(\sigma)$,

$\mathsf{harmful}(\sigma)$, resp.) to denote the dangerous (harmless, harmful) variables in the rule σ. Similarly, with $\mathsf{dang}(\Sigma)$ ($\mathsf{harmless}(\Sigma)$, $\mathsf{harmful}(\Sigma)$, resp.) we denote the sets of dangerous (harmless, harmful) variables over the entire set Σ of TGDs.

Remark 3.5. Using the previous definition, we can rephrase the Shyness conditions (see Definition 2.24) as follows:
(*Shyness Conditions*) A set Σ of TGDs is shy if, for each TGD $\sigma \in \Sigma$ the following conditions are both satisfied:

(1) if a variable x occurs in more than one body atom, then x is harmless;

(2) for every pair of distinct dangerous variable z and w in different atoms, $\mathsf{dang}(z) \cap \mathsf{dang}(w) = \emptyset$.

In order to define the notion of Dyadic TGDs, we now introduce the concept of *head-ground* set of rules, i.e., non-recursive rules in which nulls are neither created or propagated.

Definition 3.6. Let Σ a set of TGDs. A subset Σ' of Σ is called **head-ground** w.r.t Σ if:

(1) Σ' contains only Datalog rules,

(2) each head atom of Σ' contains only harmless variables w.r.t. Σ,

(3) $\mathsf{hp}(\Sigma') \cap \mathsf{bp}(\Sigma') = \emptyset$,

(4) $\mathsf{hp}(\Sigma') \cap \mathsf{hp}(\Sigma \setminus \Sigma') = \emptyset$.

The following example is given to better understand the above definition.

Example 3.7. Consider the following set of rules:

$$\begin{array}{lrcl}
\sigma_1: & R(x_1,y_1) & \rightarrow & \exists\, z_1,w_1\ Q(z_1,w_1) \\
\sigma_2: & C(y_2),R(x_2,z_2) & \rightarrow & S(y_2,z_2) \\
\sigma_3: & D(y_3,z_3),R(x_3,w_3) & \rightarrow & T(x_3,y_3) \\
\sigma_4: & Q(x_4,y_4) & \rightarrow & \exists\, z_4 A(x_4,z_4) \\
\sigma_5: & A(x_5,z_5),D(y_5,z_5) & \rightarrow & Q(x_5,y_5)
\end{array}$$

We have that a subset of head ground rule w.r.t. Σ is given by $\Sigma_{\text{HG}} = \{\sigma_2, \sigma_3\}$. In fact, since we have that $\mathsf{harmless}(\Sigma) = \{x_1, y_1, y_2, x_2, z_2, x_3, y_3, z_3, y_5, z_5\}$, it is easy to check that the head atoms of σ_2 and σ_3 contain only harmless variables and the predicates do not occur in any body of Σ_{HG}, nor in the head of rules σ_1, σ_4 and σ_5. To the contrary, rules σ_1, σ_4 and σ_5 could not be in Σ_{HG}, since they violate condition (2) and (3) of Definition 3.6. Hence, we observe that the set Σ_{HG} is maximal.

We remark that Σ_{HG} can be seen as a set of CQs. This fact will be useful in the next section. Before to define a *dyadic decomposition*, we recall the following definition.

Definition 3.8. Let $\mathbf{S}$ be a schema, and Σ_1, Σ_2 two ontologies. Then, Σ_1 and Σ_2 are **S-equivalent** (in symbols $\Sigma_1 \equiv_{\mathbf{S}} \Sigma_2$) if, for each D, q over $\mathbf{S}$, it holds that $D \cup \Sigma_1 \models q \Leftrightarrow D \cup \Sigma_2 \models q$.

Definition 3.9. Consider a class $\mathcal{C}$ of TGDs, and a set Σ of TGDs. Let $\mathbf{S} = \mathsf{sch}(\Sigma)$. A pair $(\Sigma_{\text{HG}}, \Sigma_{\mathcal{C}})$ of TGDs is a **dyadic decomposition** of Σ w.r.t. $\mathcal{C}$ if:

(1) $\Sigma_{\text{HG}} \cup \Sigma_{\mathcal{C}} \equiv_{\mathbf{S}} \Sigma$

(2) $\Sigma_{\mathcal{C}} \in \mathcal{C}$

(3) Σ_{HG} is head-ground w.r.t. $\Sigma_{\text{HG}} \cup \Sigma_{\mathcal{C}}$

(4) the head atoms of Σ_{HG} do not occur in Σ.

Dyadic-$\mathcal{C}$ is the class of all sets of TGDs that admit a dyadic decomposition w.r.t. $\mathcal{C}$.

Example 3.10. Let consider the following set Σ of TGDs.

$$\begin{array}{lrcl}
\sigma_1: & P(x_1) & \to & \exists\, y_1\; Q(x_1,y_1) \\
\sigma_2: & Q(x_2,y_2) & \to & R(y_2) \\
\sigma_3: & P(x_3) & \to & \exists\, y_3\; S(x_3,y_3) \\
\sigma_4: & S(x_4,y_4) & \to & T(y_4) \\
\sigma_5: & R(x_5),T(x_5) & \to & U(x_5)
\end{array}$$

A dyadic decomposition of Σ with respect to the class Shy is given by the pair $(\Sigma_{\mathrm{HG}},\Sigma_{\mathcal{S}})$, where, in particular, $\Sigma_{\mathrm{HG}} = (\rho'_1,\ldots,\rho'_5)$ and $\Sigma_{\mathcal{S}} = (\rho''_1,\ldots,\rho''_5)$

$$\begin{array}{lrcl}
\rho'_1: & P(x_1) & \to & Aux_1(x_1) \\
\rho'_2: & Q(x_2,y_2) & \to & Aux_2(x_2) \\
\rho'_3: & P(x_3) & \to & Aux_3(x_3) \\
\rho'_4: & S(x_4,y_4) & \to & Aux_4(x_4) \\
\rho'_5: & R(x_5),T(x_5) & \to & Aux_5(x_5)
\end{array}$$

$$\begin{array}{lrcl}
\rho''_1: & Aux_1(x_1) & \to & \exists\, y_1\; Q(x_1,y_1) \\
\rho''_2: & Aux_2(x_2),Q(x_2,y_2) & \to & R(y_2) \\
\rho''_3: & Aux_3(x_3) & \to & \exists\, y_3\; S(x_3,y_3) \\
\rho''_4: & Aux_4(x_4),S(x_4,y_4) & \to & T(y_4) \\
\rho''_5: & Aux_5(x_5) & \to & U(x_5)
\end{array}$$

According to Definition 3.9, it trivially follows the next statement.

Proposition 3.11. *Let $\Sigma \in \mathcal{C}$, for some decidable class $\mathcal{C}$ and let $(\Sigma_{\mathrm{HG}},\Sigma_{\mathcal{C}})$ its dyadic decomposition. Then*

$$\big(D \cup \Sigma\big) \models q \Leftrightarrow \big(D \cup (\Sigma_{\mathrm{HG}} \cup \Sigma_{\mathcal{C}})\big) \models q.$$

In order to prove the main result of the chapter and highlight the principal characteristic of our new fragment, we need to consider the notion of *chase bottom* defined in [27].

Definition 3.12. Consider a set Σ of TGDs, a database D, and the Herbrand Base $HB(D)$ as defined in Section 1.2. We define

$$\mathsf{chase}^{\perp}(D,\Sigma) = \mathsf{chase}(D,\Sigma) \cap HB(D),$$

that is the finite set of all null-free atoms in $\mathsf{chase}(D,\Sigma)$.

In our context, we want to restrict the chase bottom to a given set of predicates.

DEFINITION 3.13. Let Σ be a set of TGDs, D a database and X a set of predicates. We define

$$\mathsf{chase}^{\perp}_{X}(D,\Sigma) = \{\underline{a} \in \mathsf{chase}^{\perp}(D,\Sigma) : \mathsf{pred}(\underline{a}) \in X\}.$$

REMARK 3.14. Fix d as the number of distinct constants in D, and $\mu = \max_{P \in X} \mathsf{arity}(P)$, it follows that

$$|\mathsf{chase}^{\perp}_{X}(D,\Sigma)| \leq |X| \cdot d^{\mu}.$$

Hence, in data complexity the size of chase bottom restricted to a set X of predicates is polynomial, while in combined complexity is exponential.

In plain words, given a set $\Sigma \in \mathsf{Dyadic}\text{-}\mathcal{C}$, for some decidable class $\mathcal{C}$, the idea is to consider a dyadic decomposition of Σ w.r.t. $\mathcal{C}$, and compute all the ground atoms contained in $\mathsf{chase}^{\perp}_{X}(D,\Sigma_{\mathrm{HG}} \cup \Sigma_{\mathcal{C}})$, with $X = \mathsf{hp}(\Sigma_{\mathrm{HG}})$. In this way, we "complete" the database with all the possible auxiliary[1] ground atoms that we can derive from the rules of Σ_{HG}, in order to consider only the component $\Sigma_{\mathcal{C}}$ of the dyadic decomposition for query answering purpose.
Hence, our claim is to prove the following equivalence result.

LEMMA 3.15. *Consider a set* $\Sigma \in \mathsf{Dyadic}\text{-}\mathcal{C}$ *for some decidable class* $\mathcal{C}$ *and a dyadic decomposition* $(\Sigma_{\mathrm{HG}}, \Sigma_{\mathcal{C}})$ *of* Σ *w.r.t.* $\mathcal{C}$*. Let* $\Sigma' = \Sigma_{\mathrm{HG}} \cup \Sigma_{\mathcal{C}}$ *and* $X = \mathsf{hp}(\Sigma_{\mathrm{HG}})$*. For each database* D *and for each BCQ* q*, it holds that*

$$\big(D \cup \Sigma\big) \models q \Leftrightarrow \big(D \cup \mathsf{chase}^{\perp}_{X}(D,\Sigma') \cup \Sigma_{\mathcal{C}}\big) \models q.$$

PROOF. Let $\mathbf{S} = \mathsf{sch}(\Sigma)$. We observe that since $(\Sigma_{\mathrm{HG}}, \Sigma_{\mathcal{C}})$ is a dyadic decomposition of Σ, by Definition 3.9 it holds that $\Sigma \equiv_{\mathbf{S}} \Sigma'$. To prove the statement it is sufficient to prove that $\mathsf{ochase}(D \cup \Sigma) \models q \Leftrightarrow \mathsf{ochase}\big(D \cup \mathsf{chase}^{\perp}_{X}(D,\Sigma') \cup \Sigma_{\mathcal{C}}\big) \models q$.

[1]See property (4) of Definition 3.9

[$\Leftarrow$] Assume that $\mathsf{ochase}\Big(D \cup \mathsf{chase}^{\perp}_{X}(D,\Sigma') \cup \Sigma_{\mathcal{C}}\Big) \models q$. We want to prove that

$$\mathsf{ochase}(D \cup \mathsf{chase}^{\perp}_{X}(D,\Sigma') \cup \Sigma_{\mathcal{C}}) \subseteq \mathsf{ochase}(D \cup \Sigma).$$

To this end, it is sufficient to observe that $\Sigma_{\mathcal{C}} \subseteq \Sigma'$ and

$$\mathsf{chase}^{\perp}_{X}(D,\Sigma') \subseteq \mathsf{ochase}(D \cup \Sigma').$$

Hence, since $\Sigma' = \Sigma_{\mathrm{HG}} \cup \Sigma_{\mathcal{C}}$, $\mathsf{chase}^{\perp}_{X}(D,\Sigma') \subseteq \mathsf{ochase}(D \cup \Sigma)$ and, then, the thesis.

[$\Rightarrow$] Assume that $\mathsf{ochase}(D \cup \Sigma) \models q$. We want to prove that

$$\mathsf{ochase}(D \cup \Sigma) \subseteq \mathsf{ochase}(D \cup \mathsf{chase}^{\perp}_{X}(D,\Sigma') \cup \Sigma_{\mathcal{C}}).$$

By our assumptions and Proposition 3.11, we deduce that $D \cup \Sigma' \models q$, hence

$$\mathsf{ochase}(D \cup \Sigma') \models q.$$

Since $\mathsf{chase}^{\perp}_{X}(D,\Sigma') \subseteq \mathsf{ochase}(D \cup \Sigma')$, then $\mathsf{ochase}(D \cup \mathsf{chase}^{\perp}_{X}(D,\Sigma') \cup \Sigma') \models q$. We observe that, due to $\mathsf{chase}^{\perp}_{X}(D,\Sigma')$, the component $\Sigma_{\mathcal{C}}$ of the dyadic decomposition is the only relevant one. Hence, $\mathsf{ochase}(D \cup \mathsf{chase}^{\perp}_{X}(D,\Sigma') \cup \Sigma_{\mathcal{C}}) \models q$. ☐

3.2. Decidability

We now have all the tools for showing that the class of Dyadic TGDs is decidable. To this end, we provide the following algorithm to complete the database with all the auxiliary ground atoms, contained in $\mathsf{chase}^{\perp}_{X}(D,\Sigma_{\mathrm{HG}} \cup \Sigma_{\mathcal{C}})$, where $X = \mathsf{hp}(\Sigma_{\mathrm{HG}})$.

Algorithm 3. Database completion.

Input: A Dyadic decomposition $(\Sigma_{\mathrm{HG}}, \Sigma_{\mathcal{C}})$ of Σ w.r.t. $\mathcal{C}$, and a database D
Output: CBX $:= \mathsf{chase}^{\perp}_{X}(D,\Sigma_{\mathrm{HG}} \cup \Sigma_{\mathcal{C}})$, where $X = \mathsf{hp}(\Sigma_{\mathrm{HG}})$

```
(1) CBX := $\emptyset$;
(2) $\Delta := \emptyset$ ;
(3) foreach $q_i \in \Sigma_{\mathrm{HG}}$ do
(4)       $\Delta := \Delta \cup \mathsf{ans}(D \cup \mathrm{CBX}, \Sigma_{\mathcal{C}}, q_i)$;
```

```
(5) if ($\Delta \neq \emptyset$) then
(6)     $\text{CBX} := \text{CBX} \cup \Delta$;
(7)     goto Step 2
(8) return CBX;
```

Essentially, Algorithm 3 completes the database, adding to it all the auxiliary ground atoms that are possible to derive from rules of Σ_{HG}. Here is a semi-formal description of it. The first two steps are to initialize CBX, that represents the set $\mathsf{chase}_X^{\perp}(D, \Sigma_{\text{HG}} \cup \Sigma_{\mathcal{C}})$, where $X = \mathsf{hp}(\Sigma_{\text{HG}})$, and Δ that is a temporary set that we use to save the answers of each query $q_i \in \Sigma_{\text{HG}}$. The rest of the algorithm is an iterative procedure that, at each step, computes the answers of each query q_i of Σ_{HG} and completes the database until no more auxiliary ground atoms are produced.

Now we are ready to prove the main result of this section.

THEOREM 3.16. *Let $\mathcal{C}$ be a decidable class of TGDs. Then,* $\mathsf{Dyadic}\text{-}\mathcal{C}$ *is decidable.*

PROOF. Let $\Sigma \in \mathsf{Dyadic}\text{-}\mathcal{C}$. Hence, by Definition 3.9, Σ admits a dyadic decomposition $(\Sigma_{\text{HG}}, \Sigma_{\mathcal{C}})$. Let $\Sigma' = \Sigma_{\text{HG}} \cup \Sigma_{\mathcal{C}}$ and $X = \mathsf{hp}(\Sigma_{\text{HG}})$. We claim that Algorithm 3 always terminates and correctly constructs $\mathsf{chase}_X^{\perp}(D, \Sigma')$.

Termination. The algorithm clearly terminates because Step (3) is performed a finite number of times, and Step (7) never falls in a loop since we know that the size of $\mathsf{chase}_X^{\perp}(D, \Sigma')$ is bounded (see Remark 3.14). Hence, Algorithm 3 constructs $\mathsf{chase}_X^{\perp}(D, \Sigma')$ after $|\Sigma_{\text{HG}}| \cdot |X| \cdot d^{\mu}$ steps, where d is the number of different constants in the database and $\mu = \max_{P \in X} \mathsf{arity}(P)$.

Correctness. Let CBX be the output of Algorithm 3. We want to prove that $\text{CBX} = \mathsf{chase}_X^{\perp}(D, \Sigma')$.
Let us assume, by contradiction, that $\mathsf{chase}_X^{\perp}(D, \Sigma') \setminus \text{CBX} \neq \emptyset$. We recall that $\mathsf{chase}(D, \Sigma)$ can be decomposed in levels (see Section 1.4). Let us consider the chase levels

$$D = \mathsf{chase}^0(D, \Sigma') \subseteq \mathsf{chase}^1(D, \Sigma') \subseteq \mathsf{chase}^2(D, \Sigma') \subseteq \cdots,$$

where each $\mathsf{chase}^i(D,\Sigma')$ is obtained during the chase, due to atoms with maximum level $i-1$.
Let j be the minimum step in the sequence such that

$$\left(\mathsf{chase}^{\perp}_{X}(D,\Sigma')\cap\mathsf{chase}^{j-1}(D,\Sigma')\right)\subseteq \mathrm{CBX} \tag{3.1}$$

and

$$\left(\mathsf{chase}^{\perp}_{X}(D,\Sigma')\cap\mathsf{chase}^{j}(D,\Sigma')\right)\setminus \mathrm{CBX}\neq\emptyset, \tag{3.2}$$

and let $\underline{a}\in\mathsf{chase}^{j}(D,\Sigma')$ be the first atom contained in $\mathsf{chase}^{\perp}_{X}(D,\Sigma')$, but not in CBX, and generated in the chase level j. Hence, there exists a rule $\sigma\in\Sigma$ such that $\mathsf{head}(\sigma)=\underline{a}$. Considering $\mathsf{body}(\sigma)$ as a query $\tilde{q}$, we have that $\underline{a}\in\mathsf{cert}(\tilde{q},\mathsf{chase}^{j-1}(D,\Sigma'),\emptyset)$. Taking j as above, we note that

$$\mathsf{chase}^{j-1}(D,\Sigma')\subseteq\mathsf{chase}(D\cup\mathrm{CBX},\Sigma_{\mathcal{C}}).$$

By Lemma 3.15, we know that $(D\cup\Sigma')\models q\Leftrightarrow(D\cup\mathsf{chase}^{\perp}_{X}(D,\Sigma')\cup\Sigma_{\mathcal{C}})\models q$. Then, we deduce that $\underline{a}\in\mathsf{cert}(\tilde{q},\mathsf{chase}(D\cup\mathrm{CBX},\Sigma_{\mathcal{C}}),\emptyset)$. But

$$\mathsf{cert}(\tilde{q},\mathsf{chase}(D\cup\mathrm{CBX},\Sigma_{\mathcal{C}}),\emptyset)=\mathsf{cert}(\tilde{q},D\cup\mathrm{CBX},\Sigma_{\mathcal{C}}).$$

The last identity gives a contradiction with (3.2), since by our assumption $\underline{a}\notin\mathrm{CBX}$. Hence $\underline{a}$ necessary is produced by Algorithm 3.

At this point, the thesis directly follows from Lemma 3.15. $\square$

We observe that the procedure provided in Algorthm 3 does not depend on the type of chase used, hence we can use it like a "black box". The technique used to compute the answers depends on the class $\mathcal{C}$ under consideration. Taking into account that the number of steps executed by the algorithm is exponential (polynomial, resp.) in combined (data, resp.) complexity, we derive the last result of this chapter.

THEOREM 3.17. *Consider a decidable class $\mathcal{C}$ of TGDs. If* $\mathsf{CQAns}(\mathcal{C})$ *is $\circledast$-complete, for some $\circledast$ complexity class, then* $\mathsf{CQAns}(\mathsf{Dyadic}\text{-}\mathcal{C})$ *is $\circledast$-complete*

- *in data complexity if $\circledast\supseteq$*PTIME*;*

- *in combined complexity if* $\circledast \supseteq$ EXPTime *and it is* PTime-*complete in data complexity;*
- *in combined complexity if* $\circledast \supseteq i$-EXPTime *and it is* $(i-1)$-EXPTime-*complete in data complexity for* $i > 1$.

Proof. To prove the membership, it is sufficient to observe that Algorithm 3 executes a number of steps that is exponential in combined complexity, and polynomial in data complexity. Hence, we call an exponential (polynomial, resp.) number of time a procedure that, by assumption, is at least exponential (polynomial). As a result we obtain the complexity class $\circledast$. The hardness trivially follows from the fact that Dyadic-$\mathcal{C}$ includes the class $\mathcal{C}$. □

Finally, as consequence, we obtain the following:

Corollary 3.18. CQAns *over* Dyadic-Shy *and* Dyadic-Warded *is* PTime-*complete in data complexity, and* ExpTime-*complete in combined complexity.*

Proof. The proof follows from Theorem 3.17, 2.52, and 2.55. □

CHAPTER 4

The Ward^+ Class

In this chapter we present the second main result of this thesis: the new class Ward^+. This new fragment of $\text{Datalog}^\pm$ has been inspired from the combination of two existing classes: Shy and Ward, presented, respectively, in Section 2.2 and 2.4. Once we settle down its syntax, we prove that it reaches a good balance between expressivity and complexity and, moreover, we prove that this class is contained in the class Dyadic-Shy.

The chapter is structured as follows. The formal definition of the class Ward^+ is presented in Section 4.1, while the differences among the three classes cited above are outlined in Section 4.2. After that, we study the decidability and the computational complexity of Ward^+ in Section 4.3 by exploiting different technique for the proofs.

4.1. Formal Definition

In this section we introduce the new class Ward^+, derived from the combination of two existing classes: Shy and Ward. As discussed in Section 2.5, these two classes enjoy good properties as including Datalog, reaching a good compromise between expressibility and complexity, and, last but not least, have an implementation.

For the reader convenience, we recall the definition regarding the classification of variables, stated in Section 3.1.

Definition 3.1. Consider a set Σ of TGDs and a variable $z \in \mathsf{var}_\exists(\Sigma)$. A position $R[i]$ is z-affected if one between these two properties hold:

(1) there exists $\sigma \in \Sigma$ such that z appears in the head of σ at position $R[i]$;

(2) there exist $\sigma \in \Sigma$ and $x \in \mathsf{front}(\sigma)$ s.t. x occurs both in $\mathsf{head}(\sigma)$ at position $R[i]$ and in $\mathsf{body}(\sigma)$ at z-affected positions only.

A position π is **S-affected**, where $S \subseteq \mathsf{var}_\exists(\Sigma)$, if:

(1) for each $x \in S$, π is x-affected, and

(2) for each $x \in \mathsf{var}_\exists(\Sigma)$, if π is x-affected, then $x \in S$.

Definition 3.4. Given a TGD $\sigma \in \Sigma$ and a variable x in $\mathsf{body}(\sigma)$:

- if x occurs at positions $\pi_1, \ldots, \pi_n$ and $\bigcap_{i=1}^{n} \mathsf{aff}(\pi_i) = \emptyset$, then x is **harmless**,
- if x is not harmless, placed $S = \bigcap_{i=1}^{n} \mathsf{aff}(\pi_i)$, then it is **$S$-harmful**,
- if x is S-harmful and belongs to $\mathsf{front}(\sigma)$, then x is **S-dangerous**.

Every variable x that is S-dangerous (resp. S-harmful), for some $S \neq \emptyset$, is also dangerous (harmful). □

Moreover, we recall that, given a variable x that is S-dangerous, we write $\mathsf{dang}(x)$ for the set S. Given a rule $\sigma \in \Sigma$, we write $\mathsf{dang}(\sigma)$ (resp. $\mathsf{harmless}(\sigma)$, $\mathsf{harmful}(\sigma)$) to denote the dangerous (harmless, harmful) variables in the rule σ. Similarly, with $\mathsf{dang}(\Sigma)$ ($\mathsf{harmless}(\Sigma)$, $\mathsf{harmful}(\Sigma)$) we denote the sets of dangerous (harmless, harmful) variables over the entire set Σ of TGDs.

Intuitively, the syntactic condition at the base of the class Ward$^+$, can be explained as follows. Let σ be a Ward$^+$ rule. Then, $\mathsf{body}(\sigma)$ can be partitioned into two sets of atoms, B_1 and B_2, that share only harmless variables (see Figure 1). Having in mind the notion of wardedness, the set B_1 can be seen as a "multi-ward" that contains all the dangerous variables and that, at the same time, satisfies the shyness conditions. The set B_2, instead, is any atoms conjunction, that can share with B_1 only harmless variables. More formally, a set of Ward$^+$ TGDs is defined as follows.

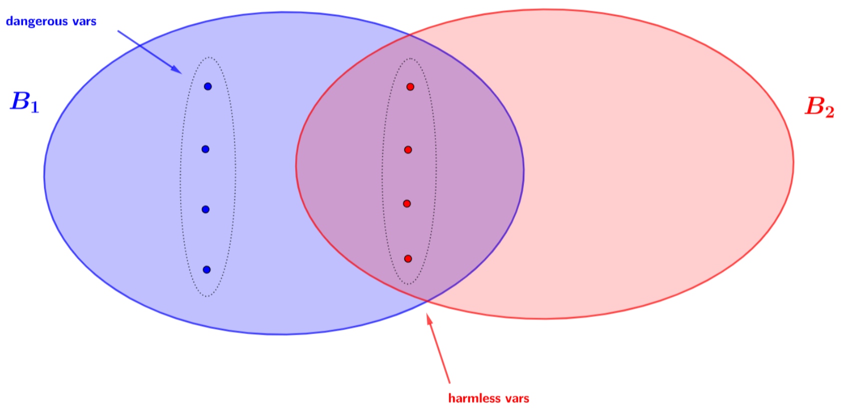


FIGURE 1. Structure of a Ward$^+$rule

DEFINITION 4.1. A set Σ of TGDs is Ward$^+$ if, for each TGD $\sigma \in \Sigma$, there are no dangerous variables in $\mathsf{body}(\sigma)$, or there exists a partition $\{B_1, B_2\}$ of $\mathsf{body}(\sigma)$ such that:

(1) B_1 contains all the dangerous variables

(2) $\mathsf{vars}(B_1) \cap \mathsf{vars}(B_2)$ are harmless variables

(3) for every pair of distinct dangerous variable z and w in different atoms, $\mathsf{dang}(z) \cap \mathsf{dang}(w) = \emptyset$

(4) for every pair of distinct atoms $\underline{a}, \underline{b} \in B_1$, $\mathsf{vars}(\underline{a}) \cap \mathsf{vars}(\underline{b})$ are harmless variables.

We write Ward^+ for the class of all finite Ward$^+$ sets of TGDs.

Now we give an example of a set of rules that belongs to Ward^+.

EXAMPLE 4.2. Consider the following set Σ of TGDs.

$$\begin{array}{lrcl}
\sigma_1: & R(x_1, y_1) & \rightarrow & \exists\, z_1\ T(z_1) \\
\sigma_2: & R(x_2, y_2) & \rightarrow & \exists\, z_2\ V(z_2) \\
\sigma_3: & S(x_3, y_3) & \rightarrow & \exists\, z_3\ P(z_3) \\
\sigma_4: & V(x_4) & \rightarrow & Q(x_4) \\
\sigma_5: & T(x_5), P(y_5), V(z_5), Q(z_5) & \rightarrow & U(x_5, y_5)
\end{array}$$

It easy to see that rules $\sigma_1, \sigma_2, \sigma_3$ and σ_4 are trivially Ward$^+$ rules w.r.t. Σ, since they are rules with one single body atom, which cannot violate any conditions of Definition 4.1. Let us focus on rule σ_5. Since $\mathsf{dang}(\sigma_5) = \{x_5, y_5\}$, $\mathsf{harmful}(\sigma_5) = \{z_5\}$ and $\mathsf{harmless}(\sigma_5) = \emptyset$, it follows that there exists a partition of $\mathsf{body}(\sigma_5)$ into two set B_1, B_2, that satisfies Definition 4.1, where, $B_1 = \{T(x_5), P(y_5)\}$ and $B_2 = \{V(z_5), Q(z_5)\}$. Hence, $\Sigma \in$ Ward$^+$.

Instead, now, we provide an example of a set of TGDs that does not belong to Ward$^+$.

Example 4.3. Consider the database $D = \{P(a)\}$, and a set Σ of rules consisting of the following rules.

$$\begin{array}{lrcl}
\sigma_1: & P(x_1) & \rightarrow & \exists\, y_1\ S(y_1) \\
\sigma_2: & S(x_2) & \rightarrow & \exists\, y_2, z_2\ R(y_2, x_2, z_2) \\
\sigma_3: & R(x_3, y_3, z_3), S(y_3) & \rightarrow & T(x_3, y_3, z_3)
\end{array}$$

The above set Σ of TGDs does not belong to Ward$^+$. In fact, variable y_3 is dangerous since it occurs in positions $R[2]$ and $S[1]$ that are affected (in particular they are both y_1-affected positions). Hence, there is a join on this dangerous variable, which contradicts condition (4) of Definition 4.1, according to which all the atoms containing dangerous variables can share only harmless variables.

4.2. Ward$^+$ vs. Shy and Ward

In this section we present the differences between the new fragment above described, Ward$^+$, and the classes Shy and Ward.

As explained in the previous section, the class Ward$^+$ is inspired to the existent classes Shy and Ward, with the aim of incorporating the benefits of both classes and building a more expressive generalization of them.

Obviously, we observe that both Shy and Ward are subset of Ward$^+$. In fact, according to Definition 4.1, the class Ward coincides trivially

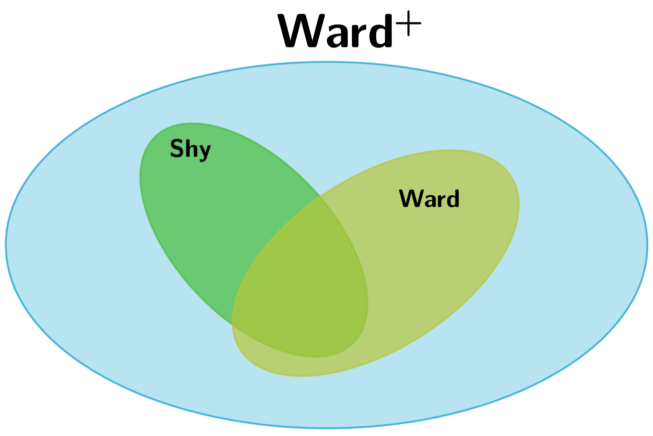


FIGURE 2. Syntactical relation among classes.

with the class Ward^+ if $|B_1| = 1$; hence, it follows that

$$\mathsf{Ward} \subseteq \mathsf{Ward}^+. \tag{4.1}$$

On the other hand, if $|B_1| > 1$ and $|B_2| = \emptyset$, we have that

$$\mathsf{Shy} \subseteq \mathsf{Ward}^+, \tag{4.2}$$

since, by Definition 4.1, the "multi-ward" satisfies the shyness conditions. However, we show that relations 4.1 and 4.2 are strictly inclusions, providing an example of a set of TGDs that belongs to Ward^+, but it is not both in Shy and Ward. To this end consider let us consider the following example.

EXAMPLE 4.4. Let us consider the following set Σ of TGDs

$$\begin{array}{lrcl}
\sigma_1 : & R(x_1, y_1) & \rightarrow & \exists\, z_1\ T(z_1) \\
\sigma_2 : & R(x_2, y_2) & \rightarrow & \exists\, z_2\ V(z_2) \\
\sigma_3 : & S(x_3, y_3) & \rightarrow & \exists\, z_3\ P(z_3) \\
\sigma_4 : & V(x_4) & \rightarrow & Q(x_4) \\
\sigma_5 : & T(x_5), P(y_5), V(z_5), Q(z_5) & \rightarrow & U(x_5, y_5)
\end{array}$$

It is easy to see that rules $\sigma_1, \sigma_2, \sigma_3$ and σ_4 are Shy, Warded and Ward$^+$ rules w.r.t. Σ, since they are rules with one single body atom, which cannot violate any condition of the classes under consideration. However, the rule σ_5 is not Ward, since the dangerous variables x_5 and y_5 are not contained in a single ward, and it does not even belong to Shy, since there is a join on the variable z_5 that is z_2-harmful. To show that the $\Sigma \in \mathsf{Ward}^+$, it is sufficient to see Example 4.2. Hence, $\Sigma \in \mathsf{Ward}^+$, but $\Sigma \notin \mathsf{Shy}$ and $\Sigma \notin \mathsf{Ward}$.

Given the above observations, we can state the following result.

THEOREM 4.5. *The classes* Shy *and* Ward *are strictly contained in the class* Ward^+*.*

REMARK 4.6. We observe that the class Ward^+ is incomparable with respect to the class *weakly frontier guarded*.

4.3. Decidability and Computational Complexity

In this section we provide the decidability of the class Ward^+ and its computational complexity. To this aim, we can exploit two different techniques:

(1) proving the existence of a dyadic decomposition;

(2) proving the existence of a well-behaved proof tree.

We start with the description of the first technique. For the reader convenience we recall the definition of the class $\mathsf{Dyadic}\text{-}\mathcal{C}$ stated in Chapter 3.

DEFINITION 3.9. Consider a class $\mathcal{C}$ of TGDs, and a set Σ of TGDs. Let $\mathbf{S} = \mathsf{sch}(\Sigma)$. A pair $(\Sigma_{\mathrm{HG}}, \Sigma_{\mathcal{C}})$ of TGDs is a **dyadic decomposition** of Σ w.r.t. $\mathcal{C}$ if:

(1) $\Sigma_{\mathrm{HG}} \cup \Sigma_{\mathcal{C}} \equiv_{\mathbf{S}} \Sigma$

(2) $\Sigma_{\mathcal{C}} \in \mathcal{C}$

(3) Σ_{HG} is head-ground w.r.t. $\Sigma_{\mathrm{HG}} \cup \Sigma_{\mathcal{C}}$

(4) the head atoms of Σ_{HG} do not occur in Σ.

$\mathsf{Dyadic}\text{-}\mathcal{C}$ is the class of all sets of TGDs that admit a dyadic decomposition w.r.t. $\mathcal{C}$.

Our aim is to show that $\mathsf{Ward}^+ \subset \mathsf{Dyadic}\text{-}\mathsf{Shy}$; hence, we want to prove the existence of a dyadic decomposition $(\Sigma_{\mathrm{HG}}, \Sigma_{\mathcal{S}})$ for every set Σ of Ward$^+$ TGDs.

Intuitively, the construction of a dyadic decomposition for a Ward$^+$ set of TGDs derives from the definition of a Ward$^+$ rule. In fact, according to Definition 4.1, a Ward$^+$ rule can be always partitioned into two sets B_1 and B_2 of atoms, where B_1 is a conjunction of atoms that satisfies the shyness conditions, while B_2 is any atom conjunction. Roughly speaking, we can use the set of atoms B_1 to build the body of a rule in $\Sigma_{\mathcal{S}}$, while, for constructing rules of Σ_{HG} we consider both B_1 and B_2, but keeping in the head only the harmless variables.

Now we give an example to clarify our idea.

Example 4.7. Consider again the set Σ of Ward$^+$ rules of Example 4.4.

$$\begin{array}{lrcl}
\sigma_1: & R(x_1,y_1) & \rightarrow & \exists\, z_1\; T(z_1) \\
\sigma_2: & R(x_2,y_2) & \rightarrow & \exists\, z_2\; V(z_2) \\
\sigma_3: & S(x_3,y_3) & \rightarrow & \exists\, z_3\; P(z_3) \\
\sigma_4: & V(x_4) & \rightarrow & Q(x_4) \\
\sigma_5: & T(x_5), P(y_5), V(z_5), Q(z_5) & \rightarrow & U(x_5,y_5),
\end{array}$$

where

$$\begin{aligned}
\mathsf{harmless}(\Sigma) &= \{x_1,y_1,x_2,y_2,x_3,y_3\},\\
\mathsf{harmful}(\Sigma) &= \{x_4,x_5,y_5,z_5\},\\
\mathsf{dang}(\Sigma) &= \{x_4,x_5,y_5\}.
\end{aligned}$$

A dyadic decomposition of Σ w.r.t. Shy is given by $(\Sigma_{\mathrm{HG}}, \Sigma_{\mathcal{S}})$, where Σ_{HG} is:

$$\begin{array}{rcl}
R(x_1,y_1) & \rightarrow & Aux_1(x_1,y_1) \\
R(x_2,y_2) & \rightarrow & Aux_2(x_2,y_2) \\
S(x_3,y_3) & \rightarrow & Aux_3(x_3,y_3) \\
V(x_4) & \rightarrow & Aux_4() \\
T(x_5), P(y_5), V(z_5), Q(z_5) & \rightarrow & Aux_5()
\end{array}$$

and $\Sigma_{\mathcal{S}}$ is:

$$\begin{array}{rcl} Aux_1(x_1,y_1) & \rightarrow & \exists\ z_1\ T(z_1) \\ Aux_2(x_2,y_2) & \rightarrow & \exists\ z_2\ V(z_2) \\ Aux_3(x_3,y_3) & \rightarrow & \exists\ z_3\ P(z_1) \\ V(x_4), Aux_4() & \rightarrow & Q(x_4) \\ V(z_5), Q(z_5), Aux_5() & \rightarrow & U(x_5,y_5) \end{array}$$

Now, we formally prove the existence of a dyadic decomposition for a Ward^{+} set Σ with respect to Shy.

THEOREM 4.8. *For every set Σ of Ward^{+} TGDs, there exists a dyadic decomposition $(\Sigma_{\mathrm{HG}}, \Sigma_{\mathcal{S}})$ of Σ with respect to* Shy.

PROOF. Let us consider a set Σ of Ward^{+} TGDs. In order to show our theorem, we provide a procedure to construct a dyadic decomposition $(\Sigma_{\mathrm{HG}}, \Sigma_{\mathcal{S}})$ of Σ with respect to Shy.

Let us consider a Ward^{+} rule

$$\sigma:\ \Phi(\mathbf{x},\mathbf{y},\mathbf{z}), \Psi(\mathbf{z},\mathbf{u}) \rightarrow \exists\ \mathbf{w}\ \Xi(\mathbf{x},\mathbf{w},\mathbf{z}),$$

where $\mathbf{x},\mathbf{y},\mathbf{z},\mathbf{u}$ are pairwise disjoint, $\Phi(\mathbf{x},\mathbf{y},\mathbf{z})$, $\Psi(\mathbf{z},\mathbf{u})$ and $\Xi(\mathbf{x},\mathbf{w},\mathbf{z})$ are conjunctions of atoms such that $\Phi(\mathbf{x},\mathbf{y},\mathbf{z}) = B_1$ and $\Psi(\mathbf{z},\mathbf{u}) = B_2$ (according to Definition 4.1). Moreover, $\mathsf{dang}(\sigma) = \{\mathbf{x}\}$, $\mathsf{harmless}(\sigma) = \{\mathbf{z}\}$ and $\mathsf{harmful}(\sigma) = \{\mathbf{x},\mathbf{u},\mathbf{y}\}$. Let $m = |\mathsf{head}(\sigma)|$, then we produce $m+2$ rules $\rho'(\sigma)$, $\rho''_0(\sigma)$, $\ldots$, $\rho''_m(\sigma)$ such that:

$$\begin{array}{rrcl} \rho'(\sigma): & \Phi(\mathbf{x},\mathbf{y},\mathbf{z}), \Psi(\mathbf{z},\mathbf{u}) & \rightarrow & Aux'_{\sigma}(\mathbf{z}) \\ \rho''_0(\sigma): & \Phi(\mathbf{x},\mathbf{y},\mathbf{z}), Aux'_{\sigma}(\mathbf{z}) & \rightarrow & \exists\ \mathbf{w}\ Aux''_{\sigma}(\mathbf{x},\mathbf{w},\mathbf{z}) \\ \rho''_1(\sigma): & Aux''_{\sigma}(\mathbf{x},\mathbf{w},\mathbf{z}) & \rightarrow & \underline{a}_1(\mathbf{v}_1) \\ \vdots & & & \\ \rho''_m(\sigma): & Aux''_{\sigma}(\mathbf{x},\mathbf{w},\mathbf{z}) & \rightarrow & \underline{a}_m(\mathbf{v}_m) \end{array}$$

where $\mathbf{v}_i \subseteq \{\mathbf{x},\mathbf{w},\mathbf{z}\}$ for each $i \in \{1,\ldots,m\}$, $\{\underline{a}_1(\mathbf{v}_1),\ldots,\underline{a}_m(\mathbf{v}_m)\} = \Xi(\mathbf{x},\mathbf{w},\mathbf{z})$ and Aux'_{σ}, Aux''_{σ} are fresh auxiliary predicates.

Now, we prove that $(\Sigma_{\mathrm{HG}}, \Sigma_{\mathcal{S}})$ is a dyadic decomposition for any Ward^{+} set of TGDs w.r.t. Shy, where

$$\Sigma_{\mathrm{HG}} = \bigcup_{\sigma \in \Sigma} \rho'(\sigma)$$

$$\Sigma_{\mathcal{S}} = \bigcup_{\substack{\sigma \in \Sigma \\ 0 \leq j \leq m}} \rho''_j(\sigma).$$

According to Definition 3.9, we have to prove four properties. We start from condition (4), observing that it is easy to see that the head predicates of Σ_{HG} do not occur in Σ, since, by construction, Aux'_σ is a fresh auxiliary predicate introduced for each $\sigma \in \Sigma$. Property (3) states that the set Σ_{HG} is head-ground w.r.t. $\Sigma_{\mathrm{HG}} \cup \Sigma_{\mathcal{S}}$. This is true since, by construction, we have that: for each $\sigma \in \Sigma$, $\rho'(\sigma)$ is a Datalog rule; $\mathsf{hp}(\Sigma_{\mathrm{HG}}) = \{Aux'_\sigma : \sigma \in \Sigma\}$, where each Aux'_σ is a predicate that does not occur neither in any body of Σ_{HG} nor in any head of $\Sigma_{\mathcal{S}}$ (i.e., $\mathsf{hp}(\Sigma') \cap \mathsf{bp}(\Sigma') = \emptyset$, and $\mathsf{hp}(\Sigma') \cap \mathsf{hp}(\Sigma \setminus \Sigma') = \emptyset$), and it contains only harmless variable. Now, we have to prove that property (2) holds, i.e., $\Sigma_{\mathcal{S}} \in \mathsf{Shy}$. This is ensured by the fact that the rule $\rho''_0(\sigma)$ is made by joining the set B_1 of σ (that has to satisfy the shyness conditions by definition), and the atom Aux'_σ, which contains only harmless variables, and hence, cannot violate any of the shyness conditions, and each rule $\rho''_j(\sigma)$, for $j = 1, \ldots, m$, is linear. Finally, property (1), that is $\Sigma_{\mathrm{HG}} \cup \Sigma_{\mathcal{S}} \equiv_{\mathsf{sch}(\Sigma)} \Sigma$, follows by construction. □

Now, we can state the following.

Theorem 4.9. $\mathsf{Ward}^{+} \subset \mathsf{Dyadic\text{-}Shy}$.

Proof. The proof directly follows from Theorem 4.8. □

Corollary 4.10. *CQAns over Ward^{+} is* PTime*-complete in data complexity, and* ExpTime*-complete in combined complexity.*

Proof. The proof follows from Theorem 4.9 and Corollary 3.18. □

Now, we continue this section with the description of the second technique, i.e., by proving the existence of a well-behaved proof tree, in order to exploit the technique developed in [23].

Despite the fact that we can deduce the complexity of Ward$^+$ by exploiting the Dyadic-Shy class, we are interested in this second technique because, thanks to this alternative methodology, we can take advantage of the Datalog rewriting algorithm deriving from proof trees. We hope to develop this idea in future works. About it, we observe that the computational complexity of Shy is ExpTime, but the existing algorithm developed for this class has complexity 2ExpTime, hence it would be interesting to investigate a possible Datalog rewriting for Ward$^+$, that would allow us a comparison with the algorithm for Ward$^+$ that we have implemented (see Chapter 5), that uses the procedure developed for Shy.

We remark that, in the following, we will focus on single-head TGDs, since we can always normalize a set of TGDs (and therefore, in particular, any Ward$^+$ set) into a single-head TGDs, preserving the certain answers. The normalization works as the one recalled in Section 2.4.

We are interested in a specific class of proof trees, for which we can bound the node-width by some polynomial function. In what follows, we give a very high-level idea about why it is possible to bound the node-width of a proof tree for a Ward$^+$ set of TGDs. In plain words, the claim is to keep small the bags of the tree, and, to this aim, we provide a bound for these bags. This bound derives from the syntactic definition of a Ward$^+$ rule. In fact, as we have explained several times in this chapter, a Ward$^+$ rule can be partitioned into two sets B_1 and B_2 of atoms which can share only harmless variables. In particular, the set of atoms B_1 satisfies the shyness conditions, while B_2 can be seen as any CQ. Hence, let $b = \max_{\sigma \in \Sigma}\{|\mathsf{body}(\sigma)|\}$, and $w = \max_{P \in \mathsf{sch}(\Sigma)}\{\mathsf{arity}(P)\}$. After resolving an atom of the query, we replace this atom with the body of some rule σ, whose size is at most b. Assuming that both $B_1, B_2 \neq \emptyset$, it is always possible to decompose this body into at least two different nodes of the chase tree, because since $\mathsf{body}(\sigma)$ is Ward$^+$, we have that $\mathsf{body}(\sigma) = B_1 \cup B_2$. However, we need to preserve the joins of the query, hence, some atoms from B_1 must be kept together the remaining atoms of q. But, from the shyness conditions, we know that are not allowed joins among nulls in $\mathsf{body}(\sigma)$; hence, of course it is possible to split in singleton the $|B_1|$ atoms. It is

necessary only to ensure the safety of the starting joins of the query. About that, exploiting again the fact that a null cannot appear in more than one atom in a shy body, in the worst case to preserve the joins explained above, it is necessary to keep together at most w atoms from B_1 with the bag containing the query atoms (while the other $|B_1| - w$ atoms can be split). Repeating this argument, we obtain the bound $|q| * w + b$.
About the set B_2, we have to observe that inside it, there could be join among harmful variables, hence it is important to preserve these joins. Intuitively speaking, this node of the chase tree becomes like a new query, hence we use the same argument presented above.
Finally, we have the following bound $\nu \cdot w + b$, with $\nu = max\{b, |q|\}$.

Now we can formally say that we are interested in a specific class of proof trees for which we can bound the node-width by the following function

$$f_{\mathrm{W}^+}(q, \Sigma) := \nu \cdot w + b,$$

where $\Sigma \in \mathsf{Ward}^+$, $b = \max_{\sigma \in \Sigma}\{|\mathsf{body}(\sigma)|\}$, $w = \max_{P \in \mathsf{sch}(\Sigma)}\{\mathsf{arity}(P)\}$, q a CQ and $\nu = \max\{b, |q|\}$.

Our aim is to prove the following result.

Theorem 4.11. *Consider a database D, a set Σ of Ward$^+$ TGDs, a CQ $q(\mathbf{x})$ and a tuple $\mathbf{c} \in \mathsf{dom}(D)^{|\mathbf{x}|}$. Let $\Sigma' = \mathcal{N}_{sh}(\Sigma)$. Then the following are equivalent:*

(1) $\mathbf{c} \in \mathsf{cert}(q, D, \Sigma)$.

(2) There exists a proof tree $\mathcal{P}$ of q w.r.t. Σ' with $\mathsf{ndw}(\mathcal{P}) \leq f_{\mathrm{W}^+}(q, \Sigma')$ such that $\mathbf{c} \in \mathcal{P}(D)$.

To this end, we need to recall some auxiliary notions introduced in the work of Berger et al. [24] and listed below for the reader convenience.

Definition 4.12 (*Blocked Application*, [24]). Let $\underline{a} \in U(\mathcal{G}^{D,\Sigma}, \Theta)$ be a node, we write $\underline{b}_1, \ldots, \underline{b}_k \Rightarrow_{(\sigma,h)} \underline{a}$ if there is a node $\mathbf{v}$ of $\mathcal{G}^{D,\Sigma}_{\Theta}$ such that $\mu_\Theta(\mathbf{v}) = \underline{a}$ and $\mathsf{succ}_{\sigma,h}(\mathbf{v}) = \underline{b}_1, \ldots, \underline{b}_k$. Moreover, for a set $\Gamma \in U(\mathcal{G}^{D,\Sigma}, \Theta)$, we say that the application of $\underline{b}_1, \ldots, \underline{b}_k \Rightarrow_{(\sigma,h)} \underline{a}$ is blocked in Γ if there is labeled null occurring in $\underline{a}$ that occurs in $\Gamma \setminus \{\underline{a}\}$, but that does not occur in any of $\underline{b}_1, \dots, \underline{b}_k$.

DEFINITION 4.13 (*Depth*, [24]). Given a node $v \in \mathcal{G}_{\Theta}^{D,\Sigma}$, the *depth* of v denoted by $\mathsf{dp}(v)$, is defined inductively as follows:

$$\mathsf{dp}(v) = \max\{\mathsf{dp}(u) \mid u \text{ is a child node of } v \text{ in } \mathcal{G}_{\Theta}^{D,\Sigma}\} + 1.$$

For an atom $\underline{a} \in U(\mathcal{G}^{D,\Sigma}, \Theta)$, we define the depth of $\underline{a}$ as

$$\mathsf{dp}(\underline{a}) = \min\{\mathsf{dp}(v) \mid \mu_{\Theta}(v) = \underline{a}\}.$$

Observe that $\mathsf{dp}(\underline{a}) = 1$ iff $\underline{a}$ labels only leaf nodes in $\mathcal{G}_{\Theta}^{D,\Sigma}$. For a set of atoms $\Gamma \in U(\mathcal{G}^{D,\Sigma}, \Theta)$, the depth is defined as

$$\mathsf{dp}(\Gamma) = \max\{\mathsf{dp}(\underline{a}) \mid \underline{a} \in \Gamma\}.$$

LEMMA 4.14 ([24]). *Consider a node v of $\mathcal{G}_{\Theta}^{D,\Sigma}$ with $\mathsf{succ}_{\sigma,h}(v) = \{\underline{b}_1, \dots, \underline{b}_k\}$ for some $\sigma \in \Sigma$ and h. Then, if some labeled null occurs in $\underline{b}_i$, it either occurs also in $\mu_{\Theta}(v)$, or it does not occur in the label of any node of $\mathcal{G}_{\Theta}^{D,\Sigma}$ that is not a descendant of v.*

Now, to prove Theorem 4.11, we need to prove the following lemma. The proof uses similar arguments proposed in the work of Berger et al. [24].

LEMMA 4.15. *Consider a database D and a set $\Sigma \in \mathsf{Ward}^{+}$ of single-head TGDs. Let $w = \max_{P \in \mathsf{sch}(\Sigma)}\{\mathsf{arity}(P)\}$, $\Theta \subseteq \mathsf{chase}(D, \Sigma)$ and $\Gamma \subseteq U(\mathcal{G}^{D,\Sigma}, \Theta)$. Then there exist a chase tree C for Γ such that*

$$\mathsf{nwd}(C) \leq f_{\mathrm{W+}}(\Gamma, \Sigma).$$

PROOF. Let us consider $\Gamma = \{\underline{a}_1, \dots, \underline{a}_k\}$ for some $k > 0$. The proof is performed by induction on $\mathsf{dp}(\Gamma)$.

BASE CASE. Assume that $\mathsf{dp}(\Gamma) = 1$. By definition of depth, this implies that $\Gamma \subseteq D$. Hence, a chase tree C for Γ is trivially given by the tree composed only by the root labeled with Γ. It is easy to see that in this case $\mathsf{nwd}(C) \leq f_{\mathrm{W+}}(\Gamma, \Sigma)$.

Induction Step. Now, let us assume that $\mathsf{dp}(\Gamma) = n$ and also that there exists a chase tree C for Γ with $\mathsf{nwd}(C) \leq f_{\mathrm{W}+}(\Gamma, \Sigma)$. We want to prove that there exists another chase tree $\bar{C}$ with bounded node-width for $\mathsf{dp}(\Gamma) = n+1$. Assume that there is only one atom in Γ with depth $n+1$. Without loss of generality, we assume that $\underline{a}_1 \in \Gamma$ is such that $\mathsf{dp}(\underline{a}_1) = n+1$ and $\mathsf{arity}(\underline{a}_1) = w$. Let $\underline{b}_1, \ldots, \underline{b}_j \in U(\mathcal{G}^{D,\Sigma}, \Theta)$ such that $\mathsf{succ}_{\sigma,h}(\underline{a}_1) = \{\underline{b}_1, \ldots, \underline{b}_j\}$ for some $\sigma \in \Sigma$ and some homomorphism h. Moreover, since $\underline{a}_1$ is of maximal depth, it follows that the application of $\underline{b}_1, \ldots, \underline{b}_j \Rightarrow_{(\sigma,h)} \underline{a}_1$ is not blocked, i.e., one of the following condition holds:

(i) for every labeled null occurring in $\underline{a}_1$, either it does not occur in $\Gamma \setminus \{\underline{a}_1\}$, or it appears necessarily in at least one atom among $\underline{b}_1, \ldots, \underline{b}_j$ (equivalently there are no nulls that are present in $\underline{a}_1$, yet not in any of the $\underline{b}_1, \ldots, \underline{b}_j$),

(ii) $\underline{a}_1$ does not contain any null.

Since Σ is Ward$^+$ there is a set B_1 of atoms such that all the nulls contained in $\underline{a}_1$ are also present in B_1. In particular, in the worst case, if all the terms of $\underline{a}_1$ are nulls, by definition of Ward$^+$, it follows that these nulls are contained in w different atoms of B_1 and that B_1 does not share any other nulls with any atom of $\mathsf{body}(\sigma) \setminus B_1$. In this case we set

$$\Gamma' = \Gamma \setminus \{\underline{a}_1\} \cup B_1 \quad \text{and} \quad \Gamma'' = \{\underline{b}_1, \ldots, \underline{b}_j\} \setminus B_1.$$

Otherwise, if $\underline{a}_1$ does not contain any null, i.e., $B_1 = \emptyset$, we define

$$\Gamma' = \Gamma \setminus \{\underline{a}_1\} \quad \text{and} \quad \Gamma'' = \{\underline{b}_1, \ldots, \underline{b}_j\}.$$

By Lemma 4.14, we deduce that the nulls that do not appear in $\underline{a}_1$ but that appear in some atom among $\underline{b}_1, \ldots, \underline{b}_j$, are all fresh. Moreover, since the application $\underline{b}_1, \ldots, \underline{b}_j \Rightarrow_{(\sigma,h)} \underline{a}_1$ is not blocked, we can conclude that, in both cases, the set $\{\Gamma', \Gamma''\}$ is a decomposition of Γ. Now, we have that $\mathsf{dp}(\Gamma') \leq n$ and $\mathsf{dp}(\Gamma'') \leq n$ (since we remove the unique atom $\underline{a}_1$ of maximal depth). Hence, by induction hypothesis, there exist two different chase trees C' and C'', respectively, one for Γ' and one for Γ''. Now, we can construct a chase tree $\bar{C}$ for Γ as follows:

(1) the root v_0 of $\bar{C}$ is labeled with Γ;

(2) v_0 has only one child v_1, whose label is $\Gamma' \cup \Gamma''$;

(3) v_1 has exactly two children, v' and v'', that are labeled respectively with Γ' and Γ''.

Finally, $|\Gamma' \cup \Gamma''| \leq |\Gamma| + \max\{|\mathsf{body}(\sigma)| : \sigma \in \Sigma\} \leq f_{\mathrm{W}^+}(\Gamma, \Sigma)$. Thus, $\bar{C}$ is a chase tree for Γ with the desired bound on the node-width.

Now it remains to show that the result holds also when there is more than one atom in Γ of depth $n+1$. This can be shown by applying a subsidiary induction on this number of atoms. □

At this point we are able to prove Theorem 4.11. We point out that this proof repeats verbatim the argument used in the proof of Theorem 4.9 in [23].

Proof of Theorem 4.11. First of all we prove the (2) ⇒ (1). To this aim let us assume that there exists a proof tree $\mathcal{P}$ of q w.r.t. Σ' with $\mathsf{ndw}(\mathcal{P}) \leq f_{\mathrm{W}^+}(q, \Sigma')$ such that $\mathbf{c} \in \mathcal{P}(D)$, hence the result follows by Theorem 2.36.

Now, we prove the converse direction, i.e., (1) ⇒ (2). Let us assume that $\mathbf{c} \in \mathsf{cert}(q, D, \Sigma)$. We need to show that if $\Sigma \in \mathsf{Ward}^+$, then there exists a proof tree $\mathcal{P}$ with respect Σ with $\mathsf{ndw}(\mathcal{P}) \leq f_{\mathrm{W}^+}(q, \Sigma')$ such that $\mathbf{c} \in \mathcal{P}(D)$. By our assumptions there exists a homomorphism h such that $h(\mathsf{atoms}(q)) \subseteq \mathsf{chase}(D, \Sigma)$ and $h(\mathbf{x}) = \mathbf{c}$. Let Θ_q be the set of atoms $h(\mathsf{atoms}(q))$. Recall that there is a homomorphism h_{Θ_q} that maps Θ_q to $U(\mathcal{G}^{D,\Sigma}, \Theta_q)$. Let homomorphism $h' = h_{\Theta_q} \circ h$ be such that $h'(\mathsf{atoms}(q)) \subseteq U(\mathcal{G}^{D,\Sigma}, \Theta_q)$ and $h'(\mathbf{x}) = \mathbf{c}$. By Theorem 4.15, there exists a chase tree $\mathcal{C}$ for $h'(\mathsf{atoms}(q))$ with $\mathsf{nwd}(\mathcal{C}) \leq f_{\mathrm{W}^+}(h'(\mathsf{atoms}(q)), \Sigma)$. By Lemma 2.42 there exists a proof tree $\mathcal{P}$ of q with respect to Σ with $\mathsf{nwd}(\mathcal{P}) \leq f_{\mathrm{W}^+}(h'(\mathsf{atoms}(q)), \Sigma) \leq f_{\mathrm{W}^+}(q, \Sigma)$ such that $\mathbf{c} \in \mathcal{P}(D)$ and the claim follows. □

Now we have another tool for showing that CQ answering under Ward$^+$ set of TGDs is in ExpTime in combined complexity and PTime in data complexity. Since our problem reduces to check if there exists a proof tree $\mathcal{P}$ of q w.r.t. Σ with $\mathsf{nwd}(\mathcal{P}) \leq f_{\mathrm{W}^+}(q, \Sigma)$ such that $\mathbf{c} \in \mathcal{P}(D)$, this can be easily checked by a space bounded algorithm that constructs, in a level-by-level fashion, the branches of the proof tree in parallel universal computations, using alternations. To this aim we exploit the technique used in [23, 24], and discussed in Section 2.5. The lower bounds are inherited from Datalog, since a set of Datalog rules (seen as TGDs) is Ward$^+$.

CHAPTER 5

Implementation and Experimental Evaluation

In this chapter we present the architecture of the prototype system regarding the class Ward^+. Specifically, we first present our approach in Section 5.1, then we describe the system used in Section 5.2 and, finally, in Section 5.3 we report and discuss some experimental results.

5.1. Algorithms for Ward^+

In this section we present the algorithm developed for the class Ward^+. We exploit the result obtained in Theorem 4.9 to use some property of Shy, including the system developed for it. The main contribution of this work consists of implementing the following algorithms:

- Null Propagation Marking Procedure (NPMP);
- Creation of the dyadic decomposition for a Ward$^+$ set of TGDs;
- BCQEval over Ward$^+$ set of TGDs.

Null Propagation Marking Procedure (NPMP)

Having in mind Definition 3.1, our first aim is to compute the set $\mathsf{aff}(\Sigma)$, that is the set containing all the affected position of an ontology Σ.

To this end, we make use of a well-known notion, the so-called *critical database* for a set of TGDs (see [69]). In plain words, the critical database is composed of all the atoms that can be formed using the predicates and the constants in the set of TGDs under consideration. If the set of TGDs is constant-free, as in our setting, then we consider an arbitrary constant $c \in \mathsf{const}(\Sigma)$. More formally, the critical database for a schema $\mathbf{S}$ is the database $D_c(\mathbf{S})$ defined as:

$$D_c(\mathbf{S}) = \{P(\mathbf{c}) \mid P \in \mathbf{S} \wedge c \in \mathsf{const}(\Sigma)\}.$$

The critical database for a set Σ of TGDs is defined as the database $D_c(\mathsf{sch}(\Sigma))$. After building the critical database, we construct a new ontology Σ' obtained by replacing every existential variable $x \in \mathsf{var}_\exists(\Sigma)$ with a fresh constant $c_x \notin \mathsf{const}(\Sigma)$.
In this way, we obtain a Datalog set of rules that, paired with the critical database, produces a model containing atoms whose terms can be only the fixed constant c, or the fresh constant c_x, for each $x \in \mathsf{var}_\exists(\Sigma)$. We outline that, since each constant c_x strictly depends on the existential variable x, we are able to collect exactly all the positions in which c_x occurs, that is, to find all the x-affected positions, for each $x \in \mathsf{var}_\exists(\Sigma)$. This is of fundamental importance for our purpose, since we need to construct the set of all the dangerous, harmful and harmless variables of a given ontology Σ. To better understand the idea behind the algorithm, we give the following example.

Example 5.1. Consider the following set Σ of TGDs:

$$\begin{array}{lrcl}
\sigma_1: & A(x_1) & \rightarrow & \exists\, y_1\ S(y_1)\\
\sigma_2: & S(x_2) & \rightarrow & \exists\, y_2, z_2\ R(y_2, x_2, z_2)\\
\sigma_3: & S(y_3), R(x_3, y_3, z_3) & \rightarrow & T(x_3, y_3, z_3)\\
\sigma_4: & T(x_4, y_4, z_4) & \rightarrow & S(z_4)
\end{array}$$

By replacing each existential variable with a fresh constant, we obtain the following set Σ' of rules:

$$\begin{array}{lrcl}
\sigma_1: & A(x_1) & \rightarrow & S(c_{y_1})\\
\sigma_2: & S(x_2) & \rightarrow & R(c_{y_2}, x_2, c_{z_2})\\
\sigma_3: & S(y_3), R(x_3, y_3, z_3) & \rightarrow & T(x_3, y_3, z_3)\\
\sigma_4: & T(x_4, y_4, z_4) & \rightarrow & S(z_4)
\end{array}$$

The model obtained from Σ' and $D_c(\mathsf{sch}(\Sigma))$ is given by:

$$\{S(1), A(1), R(1,1,1), T(1,1,1), S(c_{y_1}), R(c_{y_2}, 1, c_{z_2}), R(c_{y_2}, c_{y_1}, c_{z_2}),$$

$$T(c_{y_2}, 1, c_{z_2}), T(c_{y_2}, c_{y_1}, c_{z_2}), S(c_{z_2}), R(c_{y_2}, c_{z_2}, c_{z_2}), T(c_{y_2}, c_{z_2}, c_{z_2})\}.$$

Hence, we obtain that:

$$\mathsf{aff}(\Sigma) = \{S[1], R[1], R[2], R[3], T[1], T[2], T[3]\},$$

where:

$$\begin{array}{ccc} \mathsf{aff}(R[1]) = \{y_2\}, & \mathsf{aff}(R[2]) = \{y_1, z_2\}, & \mathsf{aff}(R[3]) = \{z_2\}, \\ \mathsf{aff}(T[1]) = \{y_2\}, & \mathsf{aff}(T[2]) = \{y_1, z_2\}, & \mathsf{aff}(T[3]) = \{z_2\}, \\ & \mathsf{aff}(S[1]) = \{y_1, z_2\}. & \end{array}$$

Once we have these information, we can easily compute the sets of dangerous, harmful and harmless variables, according to Definition 3.4.

Creation of the dyadic decomposition for a Ward^+ set of TGDs

In the following we describe the second step that we perform to construct the prototype system for Ward$^+$ sets of TGDs: given a set of TGDs $\Sigma \in \mathsf{Ward}^+$, we build a dyadic decomposition $(\Sigma_{\mathrm{HG}}, \Sigma_{\mathcal{S}})$ of Σ w.r.t. Shy. We can extract such algorithm from the proof of Theorem 4.8.

Algorithm 4. Construction of a dyadic decomposition.

Input: A set $\Sigma \in \mathsf{Ward}^+$, the set $\mathsf{aff}(\Sigma)$;
Output: A dyadic decomposition $(\Sigma_{\mathrm{HG}}, \Sigma_{\mathcal{S}})$ of Σ w.r.t. Shy.

```
(1)  $\Sigma_{\mathrm{HG}} = \emptyset$;
(2)  $\Sigma_{\mathcal{S}} = \emptyset$;
(3)  **foreach** $\sigma \in \Sigma$ **do**
(4)      $i = 0$;
(5)      $\mathrm{HL} := \mathsf{harmless}(\sigma)$;
(6)      $(B_1, B_2) := \mathit{split}(\mathsf{body}(\sigma))$;
(7)      $\mathsf{body}_{\mathrm{HG}} := \mathsf{body}(\sigma)$;
(8)      $\mathsf{head}_{\mathrm{HG}} := \mathit{Aux}_i(\mathbf{t})$, s.t. $\mathbf{t} \in \mathrm{HL}^{|\mathrm{HL}|}$;
(9)      $\mathsf{body}_{\mathrm{S}} := B_1 + \mathsf{head}_{\mathrm{HG}}$;
(10)     $\mathsf{head}_{\mathrm{S}} := \mathsf{head}(\sigma)$;
(11)     $\rho'(\sigma) := \mathit{attach}(\mathsf{body}_{\mathrm{HG}}, \mathsf{head}_{\mathrm{HG}})$;
(12)     $\rho''(\sigma) := \mathit{attach}(\mathsf{body}_{\mathrm{S}}, \mathsf{head}_{\mathrm{S}})$;
(13)     $\Sigma_{\mathrm{HG}} = \Sigma_{\mathrm{HG}} \cup \rho'(\sigma)$;
(14)     $\Sigma_{\mathcal{S}} = \Sigma_{\mathcal{S}} \cup \rho''(\sigma)$;
(15)     $i = i + 1$;
(16) **return** $(\Sigma_{\mathrm{HG}}, \Sigma_{\mathcal{S}})$;
```

BCQEval over Ward$^+$ set of TGDs

Finally we present the algorithm for the evaluation of boolean conjunctive query over a set of warded TGDs. This algorithm is based on Algorithm 3 that allows us to complete the database with all the possible ground atoms of $\mathsf{chase}^{\perp}_{X}(D, \Sigma_{\mathrm{HG}} \cup \Sigma_{\mathcal{S}})$, where $X = \mathsf{hp}(\Sigma_{\mathrm{HG}})$. However, our implementation takes advantage of the system developed for Shy. In what follows we report the pseudo-code of our approach.

Algorithm 5. BCQ evaluation over Dyadic sets of TGDs.

Input: A dyadic decomposition $(\Sigma_{\mathrm{HG}}, \Sigma_{\mathcal{S}})$, a database D and a BCQ q.

```
 (1) $n = \max_{\mathsf{body}(\sigma) \in \Sigma_{\mathrm{HG}}} \{\mathsf{join}_x \mid x \text{ is harmful}\}$;
 (2) $m = \max_{\mathsf{body}(q)} \{\mathsf{join}_x \mid x \in \mathsf{vars}(q)\}$;
 (3) $D' = \mathsf{pchase}(D \cup \Sigma_{\mathcal{S}}, n)$;
 (4) $D'' = \mathsf{pchase}(D' \cup \Sigma_{\mathrm{HG}}) \setminus D'$;
 (5) **if** $D'' \neq \emptyset$
 (6)     $D = D \cup D''$;
 (7)     **goto** step (3);
 (8) **else**
 (9)     **if** $\mathsf{pchase}(D \cup \Sigma_{\mathcal{S}}, m) \models q$
(10)         **accept**;
(11)     **else reject**;
```

5.2. System Description

Our algorithm relies on a particular Datalog engine: DLV$^{\exists}$ [1], an Answer Set Programming System that extends DLV. It was proposed by Leone et al. [63] as the first system supporting the standard first-order semantics for unrestricted CQs with existential variables, over ontologies with advanced properties. In particular, in the work of Leone et al. [64], a bottom-up evaluation strategy for shy programs was implemented inside the DLV system, followed by an improvement on the computation via some optimization techniques.

[1]https://www.mat.unical.it/dlve/

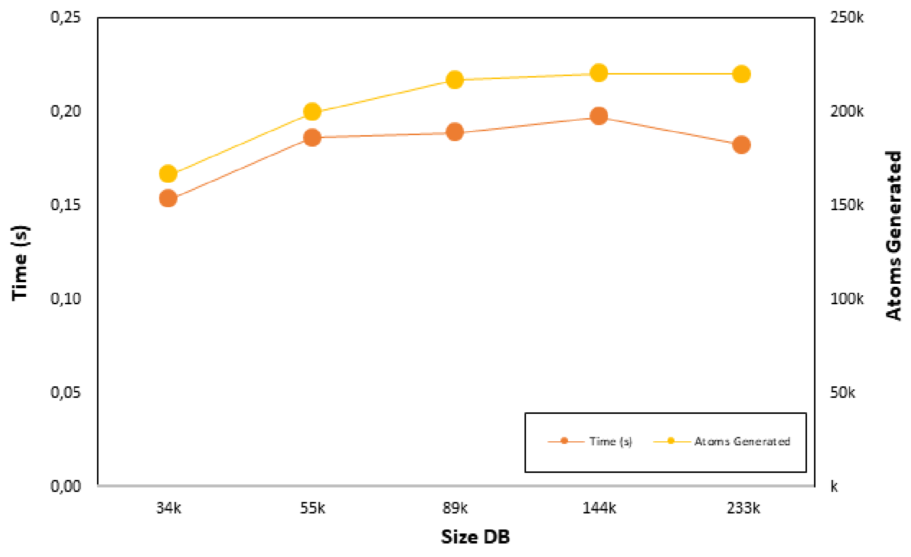
The benchmark suite proposed in the thesis belongs to the Stock Exchange domain, a real world ontology of the domain of financial institutions within the European Union, widely used in literature. It has been derived from a well-established benchmark [64]. In particular, it consists of 53 Shy (and hence Ward$^+$) axioms, 15% of which contains existential quantification, and we use five data-sets of increasing size, downloaded from `https://www.mat.unical.it/dlve`.

All tests were performed on a machine having one 2.7GHz Intel(R) Core(TM) processor and 8 GB of RAM.

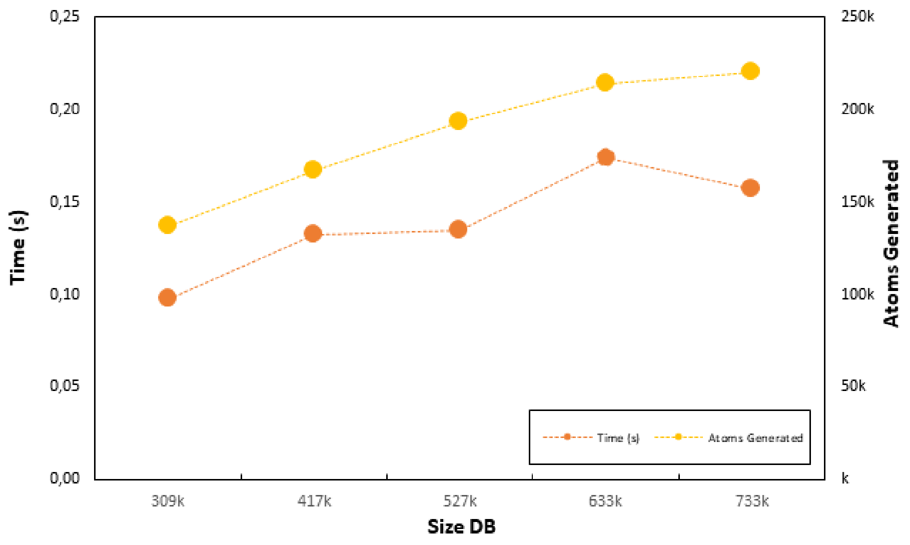
5.3. Experiments

In this section we present our experiment results. We assessed the effectiveness of our algorithm via a careful experimental activity. Specifically, we compared the query evaluation of five queries, first on the original data-set and ontology, and then on the completed database and the component $\Sigma_{\mathcal{S}}$ of the dyadic decomposition obtained from Algorithm 4. After that, we report the performance analysis of Algorithm 5 in terms of time required to complete the database, and atoms generated at each step of the completion.

Figures 1-5 show the results of the first experimental activity, i.e., starting from an ontology $\Sigma \in \mathsf{Shy}$, we compute a dyadic decomposition $(\Sigma_{\mathrm{HG}}, \Sigma_{\mathcal{S}})$ of Σ, and then we compare the evaluation of five queries over the starting database and ontology Σ (reported on the left), versus the evaluation of the queries over the completed database and $\Sigma_{\mathcal{S}}$ (reported on the right). We point out that (i) although we start from a shy ontology, it is treated by our algorithm as a true dyadic set of TGDs, since the dyadic decomposition is constructed, and hence DLV$^\exists$ is not performed in a trivial way by our algorithm, but it exploits only the component $\Sigma_{\mathcal{S}}$ of the dyadic decomposition; (ii) our algorithm produces the same answers of the standard execution and, hence, it works correctly. From these experiments, we note that the time required for query answering by the system executed on the original DB and on the completed one, are comparable.

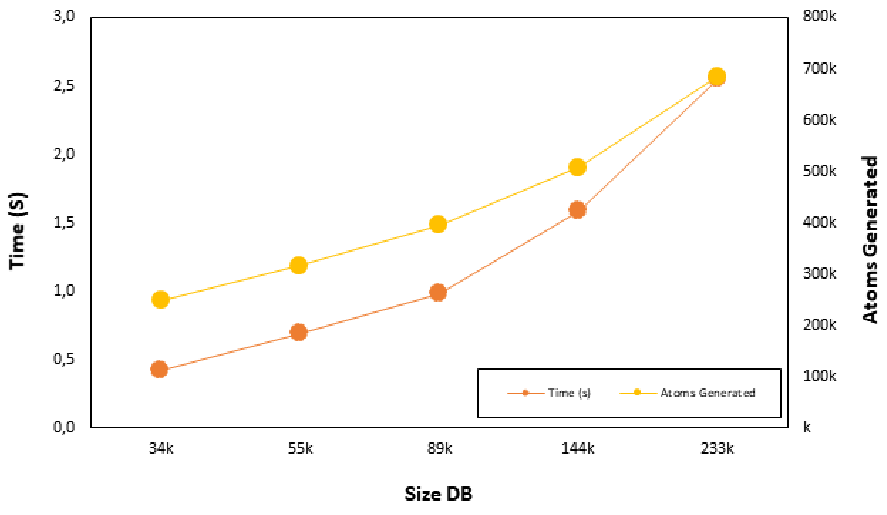


(a) Original DB

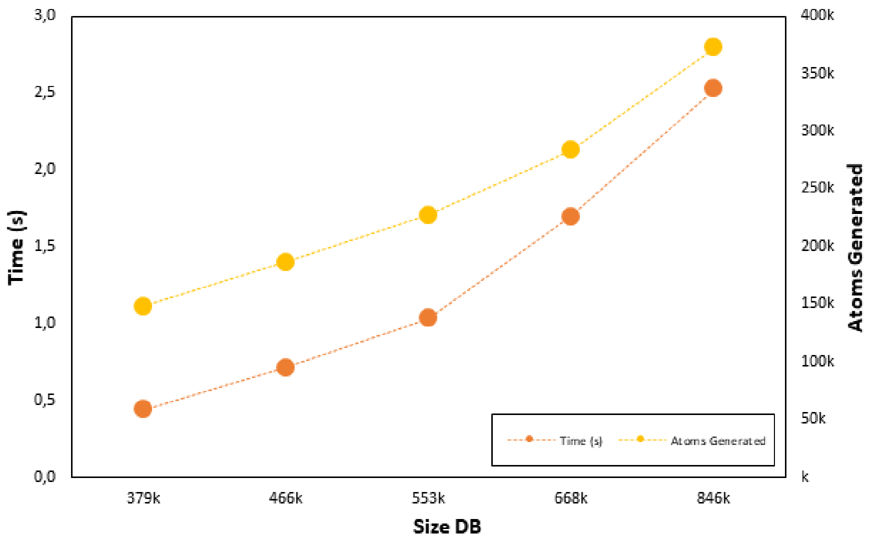


(b) Completed DB

FIGURE 1. Comparison between the evaluation of q_1 on (a) the original database and ontology, and (b) the completed database and the component $\Sigma_{\mathcal{S}}$ of the dyadic decomposition, in terms of time required (in orange) and atoms generated (in yellow).

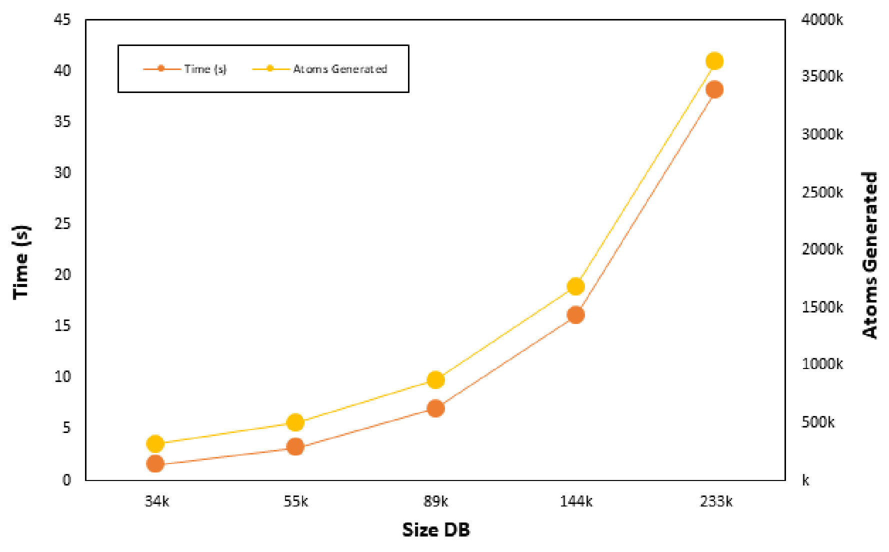


(a) Original DB

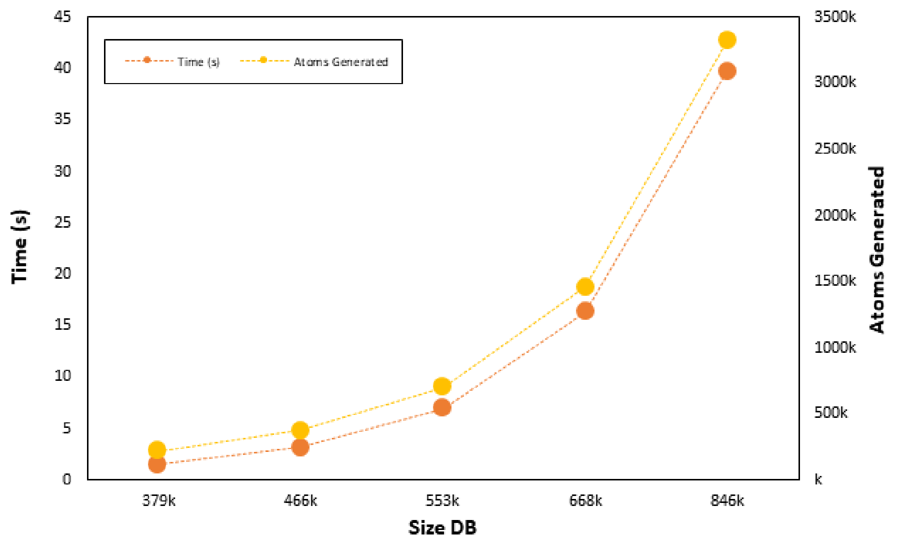


(b) Completed DB

FIGURE 2. Comparison between the evaluation of q_2 on (a) the original database and ontology, and (b) the completed database and the component $\Sigma_{\mathcal{S}}$ of the dyadic decomposition, in terms of time required (in orange) and atoms generated (in yellow).

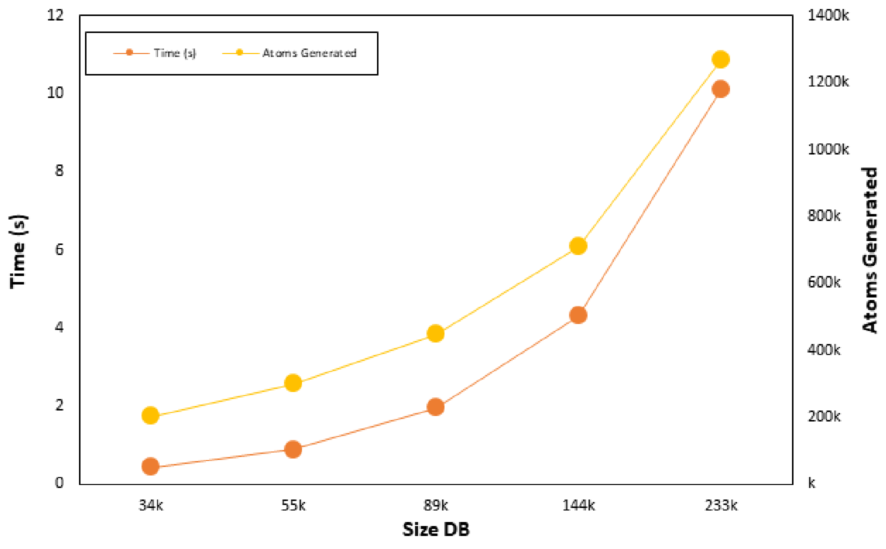


(a) Original DB

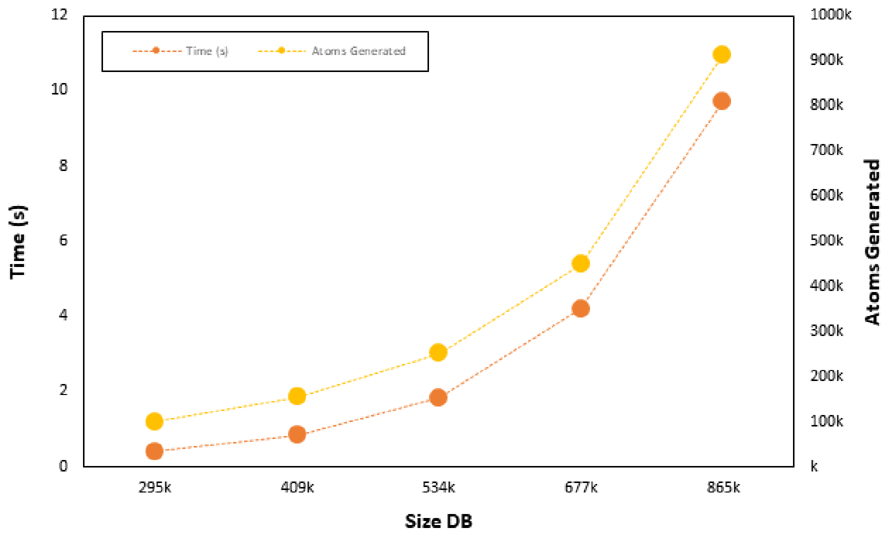


(b) Completed DB

FIGURE 3. Comparison between the evaluation of q_3 on (a) the original database and ontology, and (b) the completed database and the component Σ_S of the dyadic decomposition, in terms of time required (in orange) and atoms generated (in yellow).

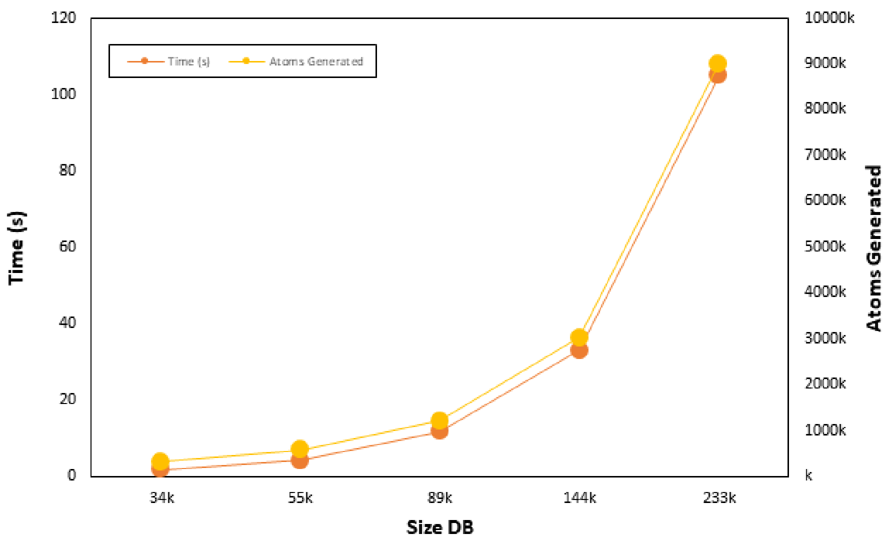


(a) Original DB

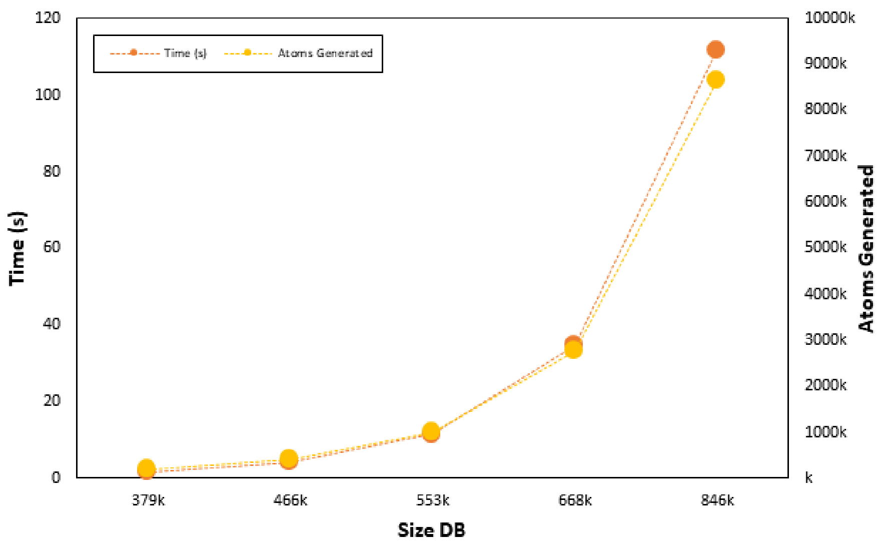


(b) Completed DB

FIGURE 4. Comparison between the evaluation of q_4 on (a) the original database and ontology, and (b) the completed database and the component Σ_S of the dyadic decomposition, in terms of time required (in orange) and atoms generated (in yellow).

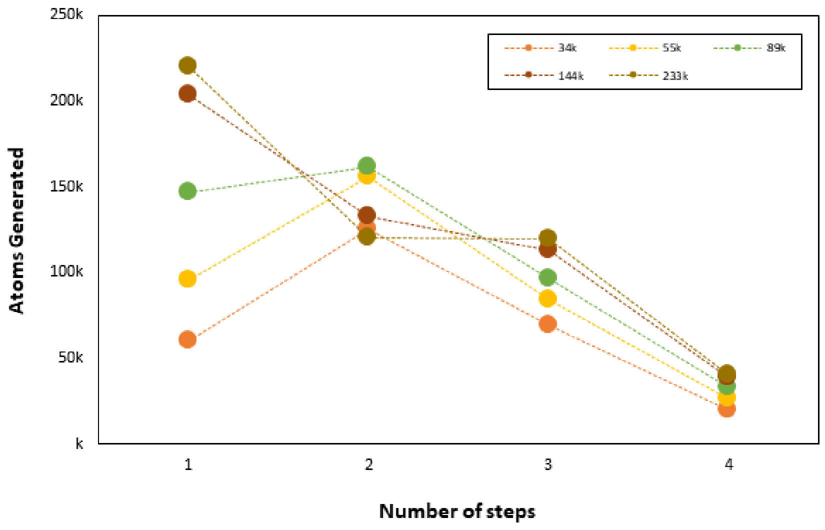


(a) Original DB

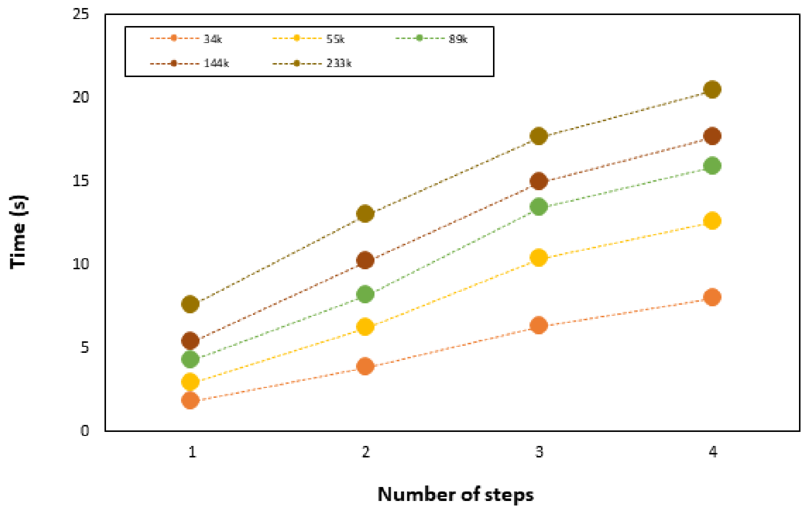


(b) Completed DB

FIGURE 5. Comparison between the evaluation of q_5 on (a) the original database and ontology, and (b) the completed database and the component Σ_S of the dyadic decomposition, in terms of time required (in orange) and atoms generated (in yellow).

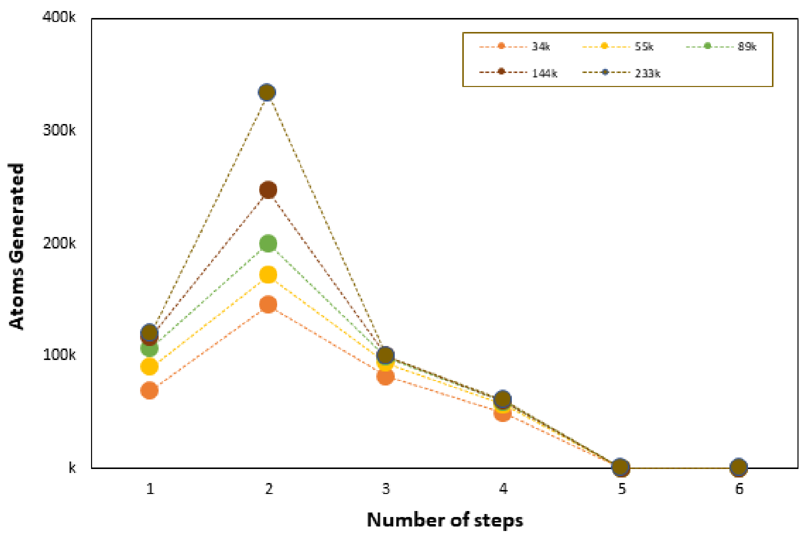


(a) Atom generated at each step

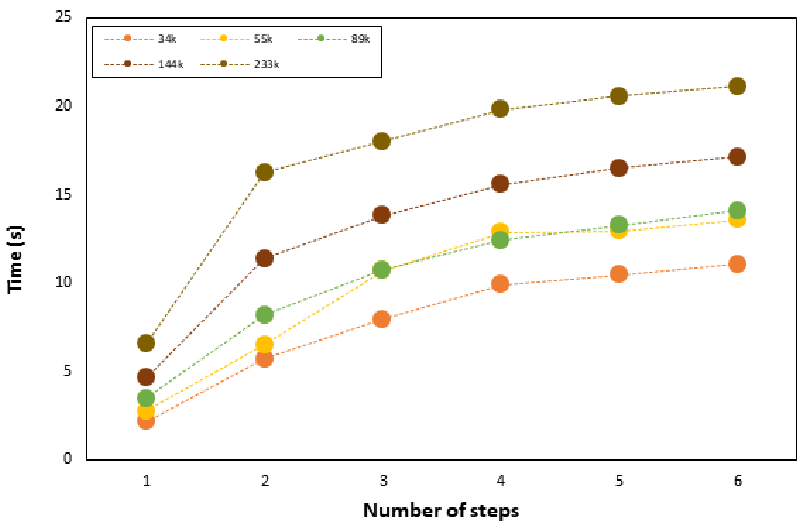


(b) Time required for each step

FIGURE 6. Graphical representation of (a) the number of atoms generated at each step of the completion of the database and (b) time necessary to perform each step for the query q_1.

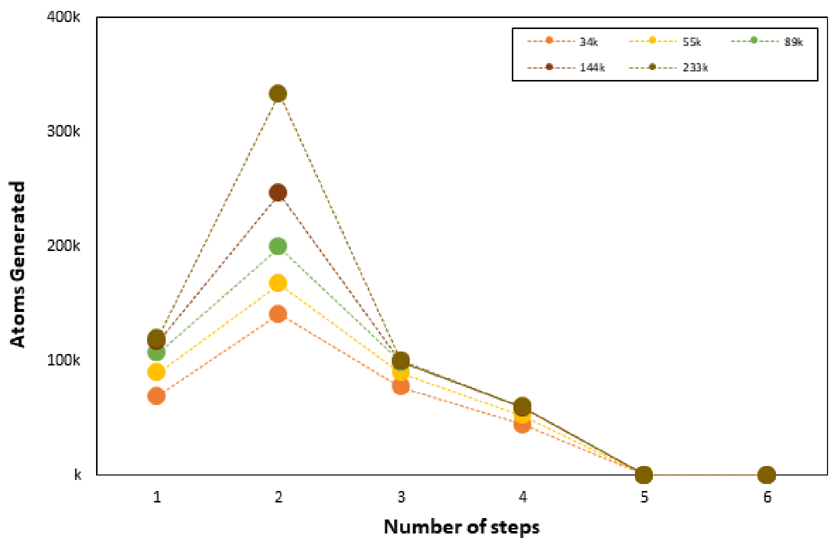


(a) Atom generated at each step

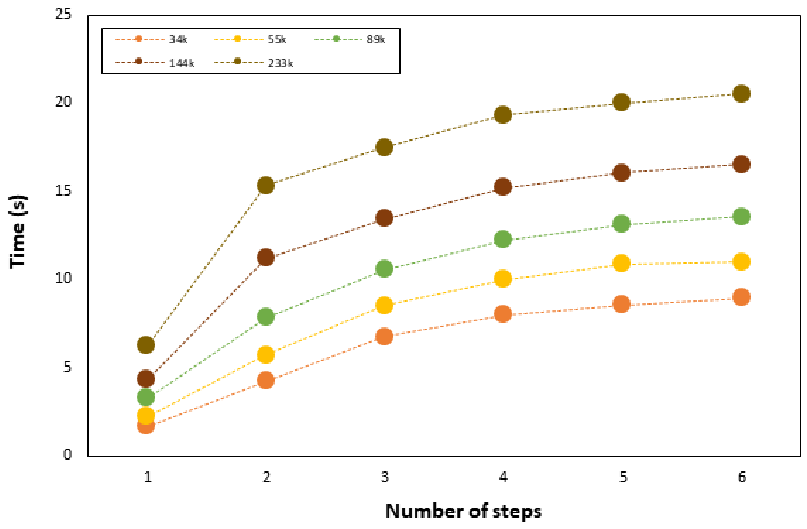


(b) Time required for each step

FIGURE 7. Graphical representation of (a) the number of atoms generated at each step of the completion of the database and (b) time necessary to perform each step for the query q_2.

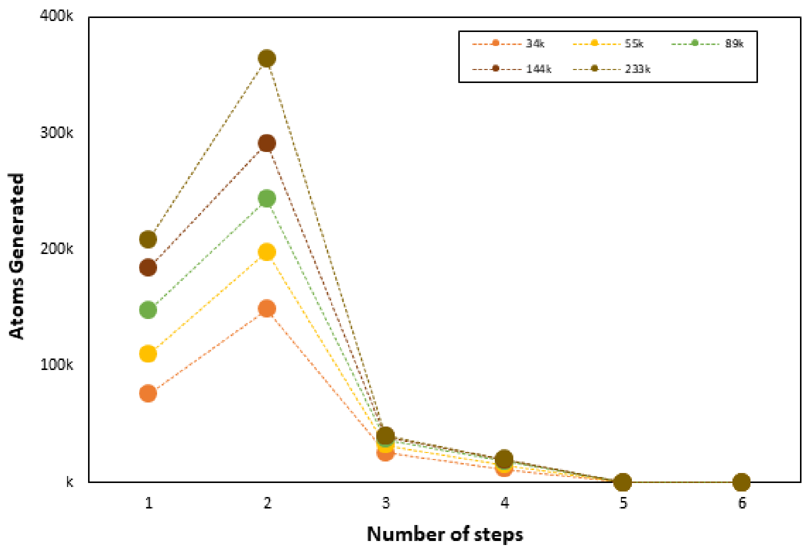


(a) Atom generated at each step

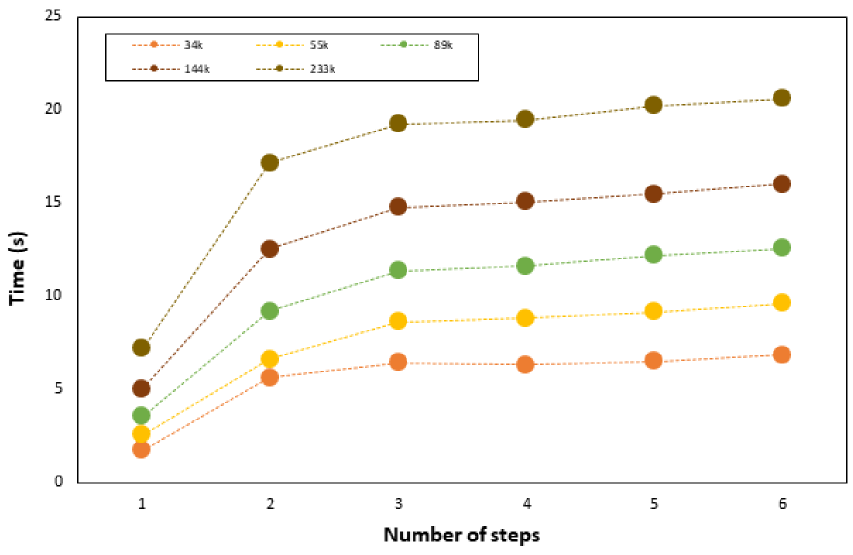


(b) Time required for each step

FIGURE 8. Graphical representation of (a) the number of atoms generated at each step of the completion of the database and (b) time necessary to perform each step for the query q_3.

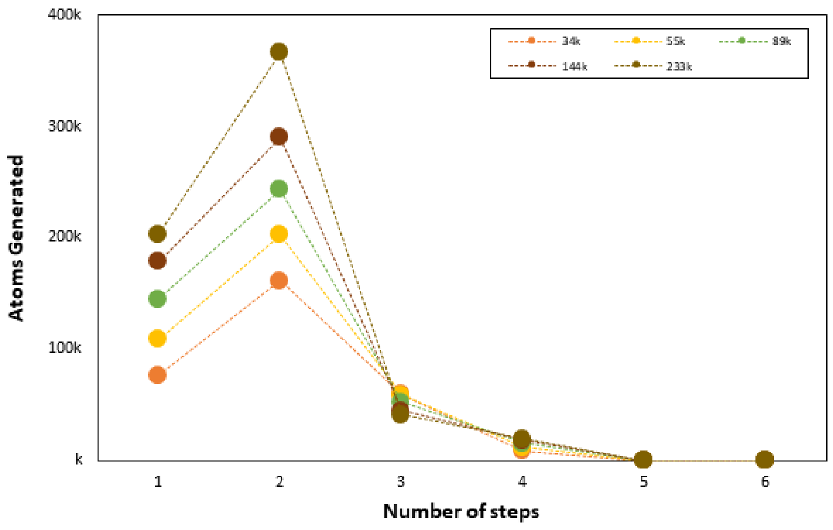


(a) Atom generated at each step

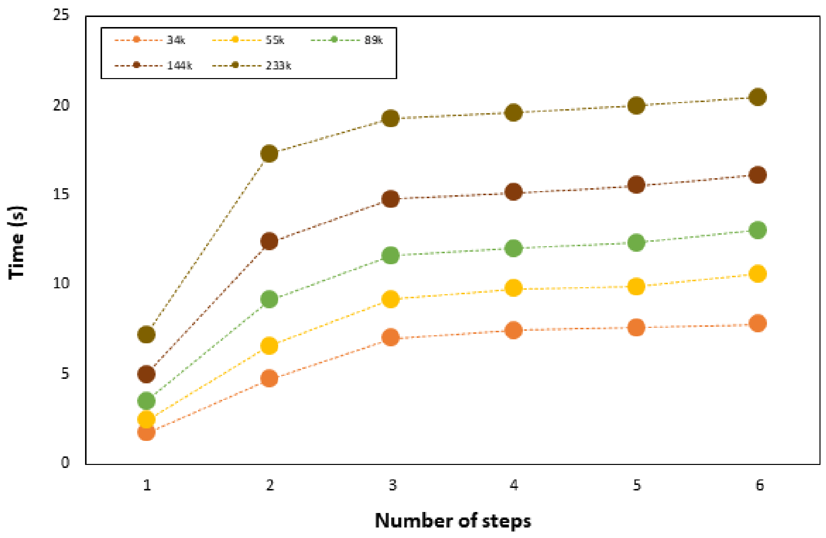


(b) Time required for each step

FIGURE 9. Graphical representation of (a) the number of atoms generated at each step of the completion of the database and (b) time necessary to perform each step for the query q_4.



(a) Atom generated at each step



(b) Time required for each step

FIGURE 10. Graphical representation of (a) the number of atoms generated at each step of the completion of the database and (b) time necessary to perform each step for the query q_5.

Figures 6-10 show the results of the second experimental activity. In particular, figures on the left show, for each step required to complete the database, the number of atoms generated by the algorithm, while figures on the right describe the time necessary to perform each step. However, we remark that there is an overhead on the time required for completing the DB, but in future projects, we aim to better understand the reason behind this overhead and to improve it with a more efficient implementation.

CHAPTER 6

Discussion and Conclusion

In recent years, query answering over ontologies (QA) has been gaining interest and getting more sought after with time especially in the data and knowledge management field. In particular, a conjunctive query (CQ) q is evaluated on a database D paired with a logical theory Σ that contains rules used to infer intentional knowledge from D. It is thus fundamental that the language with which Σ is specified should have a mix of expressiveness and complexity.

Contextually, Datalog$^{\pm}$, which is a family of languages which are Datalog-based, proposed for tractable QA, is continuously gathering interesting. Moreover, it is based on Datalog$^{\exists}$, and extension of Datalog that admits $\exists$-quantified variables in rule heads.

Nowadays, we point out that the literature is full of different classes (e.g. linear, guarded, sticky, weakly-acyclic, etc.), and it is increasingly expanding since they can be combined in various ways. But, on the other hand, not all these extensions have an implementation (e.g. tame, see [49] for more details) or, in most cases, each class requires to develop a specific solver.

In this work, we first started from the study about the combination of two existing classes, Shy and Ward, in order to exploit the systems developed for Shy. Successively, we have defined a more general class, named *Dyadic TGDs*, that allows to extend all the decidable classes, while making the most of the existent related systems. In particular, fixed a decidable class $\mathcal{C}$, Dyadic-$\mathcal{C}$ is the class of all the sets Σ of TGDs that admit a *dyadic decomposition* $(\Sigma_{\text{HG}}, \Sigma_{\mathcal{C}})$ w.r.t. $\mathcal{C}$ (i.e., there exists a rewriting of Σ in an equivalent set of TGDs given by $\Sigma_{\text{HG}} \cup \Sigma_{\mathcal{C}}$, where Σ_{HG} is a set of *head-ground* rule and $\Sigma_{\mathcal{C}} \in \mathcal{C}$). We proved that the class Dyadic-$\mathcal{C}$ is decidable, as long as $\mathcal{C}$ is decidable, providing a sound and complete algorithm used to complete the database with all the ground atoms that is possible to derive from the component Σ_{HG} of the dyadic decomposition, in order to exploit only the component $\Sigma_{\mathcal{C}}$ for query answering purpose.

Concurrently, we defined a new class of Datalog$^{\exists}$ program, called Ward^{+}, derived from the combination of Shy and Ward, since both classes enjoy good properties. We showed the complexity of this new fragment using two different techniques: we first exploited the Dyadic class and we proved that $\mathsf{Ward}^{+} \subset \mathsf{Dyadic\text{-}Shy}$; then, making use of proof trees, we confirmed the complexity class expected, getting the possibility of obtaining a Datalog rewriting for Ward^{+} in future work.

Moreover, we deepened the pchase properties, and we exploited the Bell number definition to count the exact maximal number of atoms generated by the parsimonious chase procedure. Regarding this, we conjectured the equivalence between two formulas (empirically confirmed) that we hope to formally prove in future works.

Finally, to support the results obtained in this thesis, we implemented an algorithm for the class Ward^{+}, based on the existence of a dyadic decomposition for Ward^{+} sets of TGDs, that exploits the existent system DLV$^{\exists}$, and that we hope to optimize in terms of time required to complete the database.

Bibliography

[1] Serge Abiteboul, Richard Hull, and Victor Vianu. *Foundations of databases*. Vol. 8. Addison-Wesley Reading, 1995.

[2] Alfred V Aho, Yehoshua Sagiv, and Jeffrey D Ullman. “Efficient optimization of a class of relational expressions”. In: *ACM Transactions on Database Systems (TODS)* 4.4 (1979), pp. 435–454.

[3] M. Alviano, M. Morak, and A. Pieris. “Stable model semantics for tuple-generating dependencies revisited”. In: vol. Part F127745. 2017, pp. 377–388.

[4] M. Alviano et al. “Disjunctive datalog with existential quantifiers: Semantics, decidability, and complexity issues”. In: *Theory and Practice of Logic Programming* 12.4-5 (2012), pp. 701–718.

[5] Mario Alviano and Andreas Pieris. “Default negation for non-guarded existential rules”. In: *Proceedings of the 34th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems.* 2015, pp. 79–90.

[6] Giovanni Amendola, Nicola Leone, and Marco Manna. “Finite Controllability of Conjunctive Query Answering with Existential Rules: Two Steps Forward.” In: *IJCAI.* 2018, pp. 5189–5193.

[7] Giovanni Amendola and Leonid Libkin. “Explainable Certain Answers.” In: *IJCAI.* 2018, pp. 1683–1690.

[8] Giovanni Amendola and Cinzia Marte. “Extending bell numbers for parsimonious chase estimation”. In: *European Conference on Logics in Artificial Intelligence*. Springer. 2019, pp. 490–497.

[9] Giovanni Amendola et al. “Enhancing Existential Rules by Closed-World Variables.” In: *IJCAI.* 2018, pp. 1676–1682.

[10] Giovanni Amendola et al. “Reasoning on anonymity in datalog+/-”. In: *Technical Communications of the 33rd International Conference on Logic Programming (ICLP 2017).* Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik. 2018.

[11] Marcelo Arenas, Georg Gottlob, and Andreas Pieris. “Expressive languages for querying the semantic web”. In: *ACM Transactions on Database Systems (TODS)* 43.3 (2018), pp. 1–45.

[12] Marcelo Arenas et al. *Foundations of data exchange*. Cambridge University Press, 2014.

[13] Franz Baader et al. *The description logic handbook: Theory, implementation and applications*. Cambridge university press, 2003.

[14] Jean-François Baget, Michel Leclère, and Marie-Laure Mugnier. “Walking the Decidability Line for Rules with Existential Variables.” In: *KR* 10 (2010), pp. 466–476.

[15] Jean-François Baget et al. “Extending Decidable Cases for Rules with Existential Variables.” In: *IJCAI*. Vol. 9. 2009, pp. 677–682.

[16] Jean-François Baget et al. “On rules with existential variables: Walking the decidability line”. In: *Artificial Intelligence* 175.9-10 (2011), pp. 1620–1654.

[17] Jean-François Baget et al. “Walking the complexity lines for generalized guarded existential rules”. In: *Twenty-Second International Joint Conference on Artificial Intelligence*. 2011.

[18] Vince Bárány, Georg Gottlob, and Martin Otto. “Querying the guarded fragment”. In: *2010 25th Annual IEEE Symposium on Logic in Computer Science*. IEEE. 2010, pp. 1–10.

[19] Catriel Beeri and Moshe Y Vardi. “A proof procedure for data dependencies”. In: *Journal of the ACM (JACM)* 31.4 (1984), pp. 718–741.

[20] Catriel Beeri and Moshe Y Vardi. “The implication problem for data dependencies”. In: *International Colloquium on Automata, Languages, and Programming*. Springer. 1981, pp. 73–85.

[21] Luigi Bellomarini et al. “Vadalog: a modern architecture for automated reasoning with large knowledge graphs”. In: *Information Systems* (2020), p. 101528.

[22] Luigi Bellomarini12 et al. “The VADALOG System: Swift Logic for Big Data and Enterprise Knowledge Graphs”. In: (2018).

[23] Gerald Berger et al. “The space-efficient core of Vadalog”. In: *Proceedings of the 38th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems*. 2019, pp. 270–284.

[24] Gerald Berger et al. “The space-efficient core of Vadalog”. In: *ACM Transactions on Database Systems*. Accepted for publication. 2021.

[25] Meghyn Bienvenu et al. “Ontology-based data access: A study through disjunctive datalog, CSP, and MMSNP”. In: *ACM Transactions on Database Systems (TODS)* 39.4 (2014), pp. 1–44.

[26] Pierre Bourhis et al. “Guarded-based disjunctive tuple-generating dependencies”. In: *ACM Transactions on Database Systems (TODS)* 41.4 (2016), pp. 1–45.

[27] Andrea Calì, Georg Gottlob, and Michael Kifer. “Taming the infinite chase: Query answering under expressive relational constraints”. In: *Journal of Artificial Intelligence Research* 48 (2013), pp. 115–174.

[28] Andrea Calì, Georg Gottlob, and Thomas Lukasiewicz. “A general datalog-based framework for tractable query answering over ontologies”. In: *PODS*. ACM, 2009, pp. 77–86.

[29] Andrea Calì, Georg Gottlob, and Thomas Lukasiewicz. “A general datalog-based framework for tractable query answering over ontologies”. In: *Journal of Web Semantics* 14 (2012), pp. 57–83.

[30] Andrea Calì, Georg Gottlob, and Thomas Lukasiewicz. “Datalog±: a unified approach to ontologies and integrity constraints”. In: *Proceedings of the 12th International Conference on Database Theory*. 2009, pp. 14–30.

[31] Andrea Calì, Georg Gottlob, and Andreas Pieris. “Advanced processing for ontological queries”. In: *Proceedings of the VLDB Endowment* 3.1-2 (2010), pp. 554–565.

[32] Andrea Calì, Georg Gottlob, and Andreas Pieris. “Query answering under non-guarded rules in datalog+/-”. In: *International Conference on Web Reasoning and Rule Systems*. Springer. 2010, pp. 1–17.

[33] Andrea Calì, Georg Gottlob, and Andreas Pieris. “Towards more expressive ontology languages: The query answering problem”. In: *Artificial Intelligence* 193 (2012), pp. 87–128.

[34] Andrea Calì et al. “Datalog+/-: A family of languages for ontology querying”. In: *International Datalog 2.0 Workshop*. Springer. 2010, pp. 351–368.

[35] Andrea Calì et al. "Datalog+/-: A family of logical knowledge representation and query languages for new applications". In: *2010 25th Annual IEEE Symposium on Logic in Computer Science*. IEEE. 2010, pp. 228–242.

[36] Stefano Ceri, Georg Gottlob, and Letizia Tanca. *Logic programming and databases*. Springer Science & Business Media, 2012.

[37] Stefano Ceri, Georg Gottlob, Letizia Tanca, et al. "What you always wanted to know about Datalog(and never dared to ask)". In: *IEEE transactions on knowledge and data engineering* 1.1 (1989), pp. 146–166.

[38] Ashok K Chandra and Philip M Merlin. "Optimal implementation of conjunctive queries in relational data bases". In: *Proceedings of the ninth annual ACM symposium on Theory of computing*. 1977, pp. 77–90.

[39] Surajit Chaudhuri and Moshe Y Vardi. "On the equivalence of recursive and nonrecursive datalog programs". In: *Journal of Computer and System Sciences* 54.1 (1997), pp. 61–78.

[40] C. Civili and R. Rosati. "A broad class of first-order rewritable tuple-generating dependencies". In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* 7494 LNCS (2012), pp. 68–80.

[41] Alin Deutsch, Alan Nash, and Jeff Remmel. "The chase revisited". In: *Proceedings of the twenty-seventh ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*. 2008, pp. 149–158.

[42] Ronald Fagin. "Horn clauses and database dependencies". In: *Journal of the ACM (JACM)* 29.4 (1982), pp. 952–985.

[43] Ronald Fagin et al. "Data exchange: Semantics and query answering". In: *International conference on database theory*. Springer. 2003, pp. 207–224.

[44] Ronald Fagin et al. "Data exchange: semantics and query answering". In: *Theoretical Computer Science* 336.1 (2005), pp. 89–124.

[45] Birte Glimm et al. "SPARQL 1.1 entailment regimes". In: *W3C recommendation* 21 (2013).

[46] Tomasz Gogacz and Jerzy Marcinkowski. "Converging to the Chase–A Tool for Finite Controllability". In: *2013 28th Annual ACM/IEEE Symposium on Logic in Computer Science*. IEEE. 2013, pp. 540–549.

[47] G. Gottlob, M. Manna, and A. Pieris. "Combining decidability paradigms for existential rules". In: *Theory and Practice of Logic Programming* 13.4-5 (2013), pp. 877–892.

[48] G. Gottlob et al. "Stable model semantics for guarded existential rules and description logics". In: 2014, pp. 258–267.

[49] Georg Gottlob, Marco Manna, and Andreas Pieris. "Combining decidability paradigms for existential rules". In: *Theory and Practice of Logic Programming* 13.4-5 (2013), pp. 877–892.

[50] Georg Gottlob, Giorgio Orsi, and Andreas Pieris. "Ontological queries: Rewriting and optimization". In: *ICDE*. IEEE Computer Society, 2011, pp. 2–13.

[51] Georg Gottlob, Giorgio Orsi, and Andreas Pieris. "Query rewriting and optimization for ontological databases". In: *ACM Transactions on Database Systems (TODS)* 39.3 (2014), pp. 1–46.

[52] Georg Gottlob and Christos Papadimitriou. "On the complexity of single-rule datalog queries". In: *Information and Computation* 183.1 (2003), pp. 104–122.

[53] Georg Gottlob and Andreas Pieris. "Beyond SPARQL under OWL 2 QL entailment regime: Rules to the rescue". In: *Twenty-Fourth International Joint Conference on Artificial Intelligence*. 2015.

[54] Georg Gottlob, Andreas Pieris, and Emanuel Sallinger. "Vadalog: recent advances and applications". In: *European Conference on Logics in Artificial Intelligence*. Springer. 2019, pp. 21–37.

[55] B Cuenca Grau et al. "Acyclicity notions for existential rules and their application to query answering in ontologies". In: *Journal of Artificial Intelligence Research* 47 (2013), pp. 741–808.

[56] Sergio Greco, Francesca Spezzano, and Irina Trubitsyna. "Stratification criteria and rewriting techniques for checking chase termination". In: *Proceedings of the VLDB Endowment* 4.11 (2011), pp. 1158–1168.

[57] Tomasz Imieliński and Witold Lipski Jr. “Incomplete information in relational databases”. In: *Readings in Artificial Intelligence and Databases*. Elsevier, 1989, pp. 342–360.

[58] David S Johnson and Anthony Klug. “Testing containment of conjunctive queries under functional and inclusion dependencies”. In: *Journal of Computer and system Sciences* 28.1 (1984), pp. 167–189.

[59] Mélanie König et al. “Sound, complete and minimal UCQ-rewriting for existential rules”. In: *Semantic Web* 6.5 (2015), pp. 451–475.

[60] Roman Kontchakov et al. “Answering SPARQL queries over databases under OWL 2 QL entailment regime”. In: *International Semantic Web Conference*. Springer. 2014, pp. 552–567.

[61] M. Krötzsch and S. Rudolph. “Extending decidable existential rules by joining acyclicity and guardedness”. In: 2011, pp. 963–968.

[62] Maurizio Lenzerini. “Data integration: A theoretical perspective”. In: *Proceedings of the twenty-first ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*. 2002, pp. 233–246.

[63] Nicola Leone et al. “Efficiently Computable Datalog Programs.” In: *KR* 12 (2012), pp. 13–23.

[64] Nicola Leone et al. “Fast query answering over existential rules”. In: *ACM Transactions on Computational Logic* 20.2 (2019).

[65] David Maier, Alberto O Mendelzon, and Yehoshua Sagiv. “Testing implications of data dependencies”. In: *ACM Transactions on Database Systems (TODS)* 4.4 (1979), pp. 455–469.

[66] David Maier, Yehoshua Sagiv, and Mihalis Yannakakis. “On the complexity of testing implications of functional and join dependencies”. In: *Journal of the ACM (JACM)* 28.4 (1981), pp. 680–695.

[67] Marco Manna, Francesco Ricca, and Giorgio Terracina. “Consistent query answering via ASP from different perspectives: Theory and practice”. In: *Theory and Practice of Logic Programming* 13.2 (2013), pp. 227–252.

[68] Marco Manna, Francesco Ricca, and Giorgio Terracina. “Taming primary key violations to query large inconsistent data via ASP”. In: *TPLP* 15.4-5 (2015), pp. 696–710.

[69] Bruno Marnette. “Generalized schema-mappings: from termination to tractability”. In: *Proceedings of the twenty-eighth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems.* 2009, pp. 13–22.

[70] Michael Meier, Michael Schmidt, and Georg Lausen. “On chase termination beyond stratification”. In: *arXiv preprint arXiv:0906.4228* (2009).

[71] Marie-Laure Mugnier. “Ontological Query Answering with Existential Rules”. In: *Web Reasoning and Rule Systems.* Ed. by Sebastian Rudolph and Claudio Gutierrez. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 2–23.

[72] Peter F Patel-Schneider and Ian Horrocks. “A comparison of two modelling paradigms in the semantic web”. In: *Journal of Web Semantics* 5.4 (2007), pp. 240–250.

[73] David E Smith, Michael R Genesereth, and Matthew L Ginsberg. “Controlling recursive inference”. In: *Artificial Intelligence* 30.3 (1986), pp. 343–389.