Chase-Based Ontology Mediated Query Answering: Increasing Expressiveness and Performance

by

Arash Karimi

A thesis submitted in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

Department of Computing Science

University of Alberta

Abstract

In this dissertation we consider the problem of ontology-based data access when the underlying ontology language is represented using tuple-generating dependencies (a term used in theory of databases), also known as existential rules (in the artificial intelligence literature). This problem is prominent and challenging in knowledge representation and reasoning, as it allows one to gain semantic knowledge in the presence of unknown entities that are not explicitly defined in the database. Unsurprisingly, the problem of query answering with these rules is undecidable, which means that there is no algorithm in general for which queries can be answered using finite resources. In this dissertation we build on top of the state of the art and introduce classes of rules for which the above problem is decidable with the goal of increasing expressiveness as well as improving performance.

The chase procedure provides a bottom-up method in dealing with the above problem, which comes in many variants. Toward achieving expressiveness improvements for reasoning with existential rules as the underlying ontology language, in this investigation we consider the *restricted variant* of the chase procedure. This procedure is an indispensable tool for several database applications, where its termination guarantees decidability of these tasks. Most previous studies have focused on the *Skolem chase variant* and its termination analysis. It is known that the restricted chase variant is a more powerful tool in termination analysis provided a database is given. But all-instance termination (termination for any database) presents a challenge

since the critical database and similar techniques developed for the Skolem chase variant do not work for restricted chase. In the first part of this dissertation, we develop a novel technique to characterize the activeness of all possible cycles of a certain length for the restricted chase, which leads to the formulation of a framework of parametrized class of the finite restricted chase, called k-$\mathsf{safe}(\Phi)$ rule sets. This approach is applicable to any class of finite Skolem chase identified with a condition of acyclicity. More generally, we show that the approach can be extended to the hierarchy of *bounded rule sets* previously only studied for Skolem chase. Experiments on a collection of ontologies from the web show the applicability of the proposed methods on real-world ontologies.

For the direction of performance improvements for query answering based on the chase algorithm, we study the problem of distributed reasoning over connected database components. Again, distributability over connected components is in general undecidable. We introduce the class of what we call *restricted weakly-linear disjunctive tuple-generating dependencies* as well as what is known as *generalized regular Datalog queries*. The former extends a syntactic subset of linear tuple-generating dependencies as well as that of linear disjunctive Datalog. The latter is the most expressive subset of Datalog with a decidable containment problem with which transitive closure can be expressed.

Distributed reasoning is key to unleashing the power of ontology-mediated queries constructed from these rules. We provide the first distributability results on the above classes of queries. To evaluate the practical implications of the proposed approach, we conduct experiments using real-world ontology benchmark suites. Our experimental results suggest that the performance of chase engines can be improved for query answering tasks over real-world ontologies for a subset of expressive rule languages.

Acknowledgements

Captain's Log, Stardate 2013.01. We have entered a spectacular planetary system in the Orion Arm, on the most critical mission of astrocomputational research. Our eminent guest, Arash Karimi, will attempt to study the power of knowledge representation systems.

He has a DNA based on Carbon, similar to what we found on our life readings on Earth! Fascinating! Moreover, he seems to be fascinated by the curiosity to search for unknown and to boldly go where no one has gone before. The unconventional nature of research and his passion for adventure were the reasons he was chosen to pursue a Ph.D. on planet Earth in the Solar system, which is the closest we could find in our search for extraterrestrial intelligence in the Orion Arm. His goal in this mission is to design an M6 Multitronic unit, The Ultimate Computer.

Soon after the mission started, we realized that we had chosen a wrong planet! Nobody there had a clue about M5 or even its predecessors (M1-M4). This does not seem to be the usual artificial intelligence that is taught and practiced today on planet Earth. Maybe this is the characteristics of an organism with a Carbon-based DNA to only learn and not to reason in their decision-making procedures the way we do. It then appeared to us, i.e., organisms with a Silicon-based DNA, that what seems to be a little challenging but doable in our binary-star system, is called undecidable in the Solar system!

We were there to solve and implement what we call $\aleph_{\omega_\omega}$ non-recursively enumerable class of problems, which was way beyond their reach. So, we decided to call off the mission.

But at the last minute, while visiting the Department of Computing Science at the University of Alberta, known for its world-class AI research, Arash was

lucky to find Prof. Jia You who came to the rescue. He became the captain (Arash's supervisor) to take over my role and changed the mission to make it more realistic according to Earth's human measures. Although his mission is not as ambitious as creating M6 anymore, it is ambitious in its own ways. Dr. You was one of the very few people on Earth who could help him handle the new mission successfully. He helped Arash in this critical mission in every way with his patience while listening to Arash's convoluted explanations, and many hours of supervision, writing, typesetting atomic flaws and giving greatly useful comments on various drafts. Arash couldn't have done this without his help, insight, and encouragement. Thank you, Dr. You.

This journey was as much rewarding as it was challenging and some of the related questions touch based on some of the most fundamental problems of mathematics and computer science human philosophers and mathematicians were stumbling upon. This startled Arash frequently, even though he was focused on such a teeny-tiny research area.

Arash wants to thank Dr. Heng Zhang for his useful insights and comments and to find some of the key problems which lead to his topic of research.

On a personal note, Arash would like to thank his parents and sister for their wise counsel, a sympathetic ear, patience, love, and support. Getting success in such a long mission would be impossible without them. He couldn't be luckier to have them.

Thank you Peyvand to give Arash the courage to move on with listening and advising wisely. You always inspired Arash along the way. Although you joined in the middle of the mission, it would be impossible to finish it without you.

Captain's Log, supplemental. Arash's mission to get a Ph.D. is accomplished. Good luck Dr. Karimi in your future missions.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

This dissertation is concerning an investigation of two problems.

The first one is the termination problem of the restircted chase procedure. This is given in Chapters 4 and 5 in which we introduce parametrized classes of ontology languages with finite chase which can model real-world scenarios that cannot be expressed by the known classes of finite chase in the literature.

The second one is the problem of distributed reasoning over components. This is the topic of Chapters 6 and 7 in which for the first time, we study this problem for a number of query languages and obtain promising theoretical and practical results.

This dissertation is organized as follows:

In the current chapter, we briefly introduce the research conducted in this dissertation for each of the two main topics, we provide a relatively detailed summary of the technical development.

Chapter 2 reviews some background on the theory of computational complexity relevant to our dissertation. In the rest of this thesis, we frequently refer to the materials of this chapter to establish our technical results on complexity analyses of our proposed algorithms, as well as on reasoning and data complexities.

In Chapter 3, we present the state of the art in different aspects that are affected by the results of this dissertation. In future chapters we establish our contributions on top of what is presented in this chapter.

The topic of Chapter 4 is our contribution on classes of rule sets with finite

restricted chase, which we call k-safe rule sets, along with some experiments conducted on real-world ontology benchmarks targeted at membership and chase computation analysis with k-safe(Φ_Δ) for different acyclicity conditions Δ.

Chapter 5 continues its predecessor by applying our ideas which lead to the introduction of k-safe rules to extend bounded hierarchy of ontologies with finite Skolem chase. Then, for the hierarchy introduced in this chapter, we conduct membership and reasoning complexity analyses.

In Chapter 6, we present our results on distribution over components for a class of queries we introduce in this work which are based on what we call restricted weakly-linear (RWL) rule sets. This class of rules extend a syntactic subset of linear existential tuple-generating dependencies as well as that of linear disjunctive Datalog. Experiments are conducted to assess the theoretical results on distribution over components provided in this chapter as well as the performance of chase-based query answering on distributable queries.

Chapter 7 continues the contributions of its predecessor by studying the problem of distribution over components for a class of queries constructed from what is known as generalized regular Datalog queries. In this chapter, we identify a syntactic subclass of these queries, which we call pseudo-guarded generalized Datalog queries, and show that the problem of distribution over components for these queries is decidable. This paves the way to improve the query answering performance when utilizing these queries. The experiments we conduct at the end of this chapter evaluate our theoretical contributions on real-world ontology benchmarks.

Finally, in Chapter 8, we conclude this dissertation, summarize it and show directions for the future work related to our results.

We organize the rest of this chapter as follows. In Section 1.1 we introduce the formalism of existential rules and the chase procedure, and illustrate our technical contributions of Chapters 4 and 5. Section 1.2 gives an introduction to our contributions to be reported in Chapter 6. Finally, in Section 1.3, we summarize our contributions of Chapter 7.

1.1 K-Safe and Bounded Hierarchy of Terminating Ontologies under Restricted Chase

The advent of emerging applications of knowledge representation and ontological reasoning has been the motivation of recent studies on rule-based languages, known as tuple-generating dependencies (TGDs) [24], existential rules [18] or Datalog$^{\pm}$ [45], which are considered as a powerful modelling language for applications in data exchange, data integration, ontological querying, and so on. A major advantage of using these languages is that the formal semantics based on first-order logic facilitates reasoning in an application, where answering conjunctive queries over a database extended with a set of existential rules is a primary task, but unfortunately an undecidable one in general [25]. The *chase procedure* is a bottom-up algorithm that extends a given database by applying specified rules. If such a procedure terminates, given an input database I, a finite rule set R and a conjunctive query, we can answer the query against R and I by simply evaluating it on the result of the chase. In applications such as in data exchange scenarios, we need the result that the chase terminates for all databases. Thus, determining if the chase of a rule set terminates is crucial in these applications.

Existential rules in this context are implications of the form

$$\forall \mathbf{x} \forall \mathbf{y} \ (\phi(\mathbf{x}, \mathbf{y}) \rightarrow \exists \mathbf{z} \ \psi(\mathbf{x}, \mathbf{z}))$$

where ϕ and ψ are conjunctions of atoms.

For example, that *every student has a classmate who is also a student* can be expressed by

$$\text{Student}(x) \rightarrow \exists z \ \text{Classmate}(x, z), \text{Student}(z)$$

where universal quantifiers are omitted.

We can remove existential quantifiers by Skolemization where existential variables are replaced by Skolem terms. For the above example, the resulting Skolemized rule is

$$\text{Student}(x) \rightarrow \text{Classmate}(x, f_z(x)), \text{Student}(f_z(x))$$

where f_z is a Skolem function. Given a database, say $I = \{\text{Student}(a)\}$, the atom in it triggers the application of the rule, which will first add in I the atoms $\text{Classmate}(a, f_z(a))$, $\text{Student}(f_z(a))$; repeated applications will further add $\text{Classmate}(f_z(a), f_z(f_z(a)))$, $\text{Student}(f_z(f_z(a)))$, and so on. In this example, the chase produces an infinite set.

Note that a set of Skolemized rules is a Horn logic program.

Four main variants of the chase procedure are considered in the literature, which are called *oblivious* [66], *Skolem* [113] (*semi-oblivious*),[1] *restricted* (a.k.a. *standard*) [66], and *core* chase [61].

What is common to all these chase variants is the property that, for any database instance I, a finite rule set R and a Boolean conjunctive query q, q is entailed by R and I if and only if it is entailed by the result of the chase on R and I. However, these chase variants behave differently concerning termination. The oblivious chase is weaker than the Skolem chase, in the sense that whenever the oblivious chase terminates, so does the Skolem chase, but the reverse does not hold in general. In turn, the Skolem chase is weaker than the restricted chase, which is itself weaker than the core chase.

The core chase is defined based on the restricted chase combined with the notion of *cores of relational structures* [91]. This variant of chase is theoretically interesting as it captures all universal models of a given rule set and instance.[2] Given a rule set R and an instance I, whenever there is a universal model of R and I, the core chase produces such a model.

As the cost of each step of the core chase is DP-complete, this chase variant is a bit more complicated than the other main chase variants and to the best of our knowledge, there are no known efficient algorithms to compute the core when the instances under evaluation are of nontrivial sizes.

In Chapters 4 and 5, we will be focusing on the Skolem and the restricted

[1]The chase using Skolemized rules can be expressed equivalently by introducing fresh nulls. The chase under these two different notations are considered equivalent due to a one-to-one correspondence between generated Skolem terms and introduced fresh nulls.

[2]Given an instance I and a rule set R, an instance J is a model of R and I if J satisfies all rules in R and there is a homomorphism from I to J. Moreover, a model U is universal for R and I if it has homomorphism into every model of R and I. Models of R and I are not unique, but universal models of R and I are unique up to homomorphism.

versions of the chase, which are the most investigated in the literature, as the core chase is computationally costly in practice (cf. [31] for more details).

Despite the existence of many notions of acyclicity in the literature (cf. [56] for a survey), there are natural examples from real-world ontologies that are nonterminating under the Skolem chase but terminating under the restricted chase. However, finding a suitable characterization to ensure restricted chase termination is a challenging task, and in the last decade, to the best of our knowledge, only a few conditions are discovered. In [49], the classes of *restricted joint acyclicity* (RJA), *restricted model-faithful acyclicity* (RMFA), and *restricted model-summarizing acyclicity* (RMSA) are introduced for the restricted chase which generalize the corresponding classes under the Skolem chase, namely (by removing the letter R in the above names) *joint-acyclicity* (JA) [105], *model-faithful acyclicity* (MFA), and *model-summarizing acyclicity* (MSA) [56], respectively. Intuitively, the classes for the restricted chase introduce a *blocking criterion* to check if the head of each rule is already entailed by the derivations when constructing the arena for checking the corresponding acyclicity conditions for JA, MFA, and MSA, respectively. Here, we extend their work in two different directions. First, we provide a highly general theoretical framework to identify strict superclasses of all existing classes of finite Skolem chase that we are aware of, and second, we show a general critical database technique, which works uniformly for all bounded finite chase classes.

With the curiosity on the intended applications of some of the practical ontologies that we collected from the web (which will be used in our experimentation to be reported later in this chapter) and the question why the restricted chase may help identify classes of terminating rule sets, we analyze some of them to get an understanding. Here, let us introduce a case study of policy analysis for access control, which is abstracted from a practical ontology from the considered collection. This example shows how the user may utilize the approach we develop in this chapter to model and reason with a particular access policy.

Consider a scenario involving several research groups in a given lab located in a department. Each one of these groups may have some personnel working

in labs. Also, each person may possess keys which are access cards to the labs of that department. The set of rules $R = \{r_1, r_2, r_3, r_4, r_5\}$ below is intended to model the access policy to the labs: any member of any research lab must be able to enter his or her lab that is assigned to the research group (r_1); for each person x who has a key to a room y, there is a lab u such that x can enter u and the key y opens the door of that lab (r_2); and if a person can enter a lab, he or she must have a matching key that opens the lab (r_3).

An employee of the department is responsible for granting the keys to labs (r_4). Once an employee grants a key to a person, the grantee is assumed to be in the possession of the key (r_5).

$$
\begin{aligned}
&r_1 : \mathrm{MemOf}(x, y) \rightarrow \mathrm{Enters}(x, y)\\
&r_2 : \mathrm{HasKey}(x, y) \rightarrow \exists u\ \mathrm{Enters}(x, u), \mathrm{KeyOpens}(y, u)\\
&r_3 : \mathrm{Enters}(x, y) \rightarrow \exists v\ \mathrm{HasKey}(x, v), \mathrm{KeyOpens}(v, y)\\
&r_4 : \mathrm{HasKey}(x, y) \rightarrow \exists w\ \mathrm{Grants}(w, x, y), \mathrm{Emp}(w)\\
&r_5 : \mathrm{Grants}(t, x, y) \rightarrow \mathrm{HasKey}(x, y)
\end{aligned}
$$

The intended meanings of the predicates are: $\mathrm{MemOf}(x, y)$ represents that x is a member of (lab) y; $\mathrm{Enters}(x, y)$ says that (person) x enters (lab) y; $\mathrm{HasKey}(x, y)$ affirms that (person) x has a key card to (room) y; $\mathrm{KeyOpens}(y, u)$ means that the key to (room) y opens (lab) u; Furthermore, by $\mathrm{Grants}(w, x, y)$, we declare that (employee) w grants (person) x access to (room) y; finally, $\mathrm{Emp}(w)$ confirms that w is an employee of the department.

The rules in R can be applied cyclically. For example, an application of r_4 triggers an application of r_5 which triggers r_4 again. But even under the Skolem chase variant, these two rules do not produce an infinite derivation sequence. Let us consider the path $\pi_1 = (r_4, r_5)$ (a path is just a sequence of rules) and show the Skolem chase derivations of $sk(\pi_1)$ (the Skolemized version of rules in π) from $\{\mathrm{HasKey}\big(a, b\big)\}$. Recall that the Skolem chase considers the Skolemized version of the rules. In the following, Skolem functions are written as f_z for existential variable z and $\xrightarrow{\langle sk(r), \tau\rangle}$ denotes that rule $sk(r)$ is applied using substitution τ.

$$
\begin{aligned}
&I_0 = \{\mathrm{HasKey}\big(a, b\big)\} \xrightarrow{\langle sk(r_4), \{x/a, y/b\}\rangle}\\
&I_1 = I_0 \cup \big\{\mathrm{Grants}\big(f_w(a, b), a, b\big), \mathrm{Emp}\big(f_w(a, b)\big)\big\} \xrightarrow{\langle sk(r_5), \{t/f_w(a,b), x/a, y/b\}\rangle}\\
&I_2 = I_1
\end{aligned}
$$

The sequence of derivations for the path $\pi_2 = (r_5, r_4)$ can be obtained similarly. From these derivations, we can observe that any path of rules that only consist of r_4 and r_5 is terminating under the Skolem chase.

However, the cyclic applications of r_2 and r_3 lead to an infinite Skolem chase. To illustrate, let us construct a Skolem chase sequence starting from the application of r_2 on a singleton database $\{\text{HasKey}(a, b)\}$ as follows (where the existential variable u in r_2 is Skolemized to $f_u(x, y)$ and v in r_3 is Skolemized to $f_v(x, y)$):

$$
\begin{aligned}
&I_0 = \{\text{HasKey}\big(a, b\big)\} \xrightarrow{\langle sk(r_2), \{x/a, y/b\}\rangle} \\
&I_1 = I_0 \cup \big\{\text{Enters}\big(a, f_u(a, b)\big), \text{KeyOpens}\big(b, f_u(a, b)\big)\big\} \xrightarrow{\langle sk(r_3), \{x/a, y/f_u(a,b)\}\rangle} \\
&I_2 = I_1 \cup \big\{\text{HasKey}\big(a, f_v(a, f_u(a, b))\big), \text{KeyOpens}\big(f_v(a, f_u(a, b)), f_u(a, b)\big)\big\} \\
&\xrightarrow{\langle sk(r_2), \{x/a, y/f_v(a, f_u(a,b))\}\rangle} \\
&I_3 = I_2 \cup \big\{\text{Enters}\big(a, f_u(a, f_v(a, f_u(a, b)))\big), \\
&\qquad\qquad \text{KeyOpens}\big(f_v(a, f_u(a, b)), f_u(a, f_v(a, f_u(a, b)))\big)\big\} \\
&\ldots
\end{aligned}
$$

On the other hand, in each valid derivation of a restricted chase sequence, we must ensure that each rule r_i that is used in the derivation is not already satisfied by the current conclusion set, which is the set of all derivations generated so far right before application of r_i.

Though the Skolem chase leads to an infinite sequence, the restricted chase does terminate. Utilizing fresh nulls, denoted by n_i, for the representation of unknowns,[3] we have the following sequence of restricted chase derivations for this rule set, where θ is a substitution which maps n_3 to n_1 and other symbols to themselves. From this derivation sequence, it can be seen that I_3 is not a new instance, and therefore, (r_2, r_3, r_2) is not an active path, i.e., the one that leads to a (valid) restricted chase sequence.

$$
\begin{aligned}
&I_0 = \{\text{HasKey}\big(a, b\big)\} \xrightarrow{\langle r_2, \{x/a, y/b\}\rangle} \\
&I_1 = I_0 \cup \big\{\text{Enters}\big(a, n_1\big), \text{KeyOpens}\big(b, n_1\big)\big\} \xrightarrow{\langle r_3, \{x/a, y/n_1\}\rangle} \\
&I_2 = I_1 \cup \big\{\text{HasKey}\big(a, n_2\big), \text{KeyOpens}\big(n_2, n_1\big)\big\} \xrightarrow{\langle r_2, \{x/a, y/n_2\}\rangle} \\
&I_3 = I_2 \cup \big\{\text{Enters}\big(a, n_3\big), \text{KeyOpens}\big(n_2, n_3\big)\big\} \xRightarrow{\theta=\{n_3/n_1\}} \theta(I_3) \subseteq I_2
\end{aligned}
$$

[3]For the clarity of illustration, we use fresh nulls instead of Skolem terms – there is a one-to-one correspondence between these two kinds of representations of unknown elements.

From the above sequence of derivations, it can be seen that when we attempt to apply r_2 on I_2, its head can be instantiated to Enters$(a, _)$ and KeyOpens$(n_2, _)$, where we place an underline to mean that the existential variable v in r_3 can be instantiated to form atoms that are already in I_2, which halts the derivation under the restricted chase.

In Chapter 4, we will show that we can run such tests on cyclic rule applications of a fixed nesting depth, which we call k-cycles ($k > 0$), with the underlying databases, which we call *restricted critical databases*, to define a hierarchy of classes of the finite restricted chase.

In addition, we show how to extend δ-*bounded ontologies*, which were introduced in the context of the Skolem chase variant [144], uniformly to δ-bounded rule sets under the restricted chase variant, where δ is a bound function for the maximum depth of chase terms in a chase sequence. Furthermore, as a concrete case of δ, we consider functions constructed from an exponential tower of the length κ (called exp_κ in Chapter 5), for some given integer κ, and then we obtain the membership as well as reasoning complexities with these rule sets.

The main contributions of Chapters 4 and 5 are as follows:

1. In termination analysis for the Skolem chase, a major advance is the so called critical database technique [113], which says that the termination of Skolem chase w.r.t. all databases can be faithfully simulated by termination of Skolem case w.r.t. a single database called the critical database. We show that while the traditional critical database technique does not work for the restricted chase, a kind of "critical databases" exist by which any finite restricted chase sequence can be faithfully simulated. This is shown by Theorem 33 for rules whose body contains no repeated variables (called *simple rules*) and generalized by Theorem 34 for arbitrary rules.

2. As the above results provide sufficient conditions to identify classes of the finite restricted chase, we define a hierarchy of such classes, which can be instantiated to a concrete class of finite chase, given an acyclic-

ity condition. This is achieved by Theorem 40 based on which various acyclicity conditions under the Skolem chase can be generalized to introduce classes of finite chase beyond finite Skolem chase.

3. Our experimental results on a large set of ontologies collected from the web show practical applications of our approach to real-world ontologies. In particular, in contrast with the current main focus of the field on acyclicity conditions for termination analysis, our experiments show that many ontologies in the real-world involve cycles of various kinds but indeed fall into the finite chase.

Furthermore, built on top of k-safe rules, our contributions in Chapter 5 are as follows:

1. We show that the hierarchy of δ-bounded rule sets under the Skolem chase [144] can be generalized by introducing δ-bounded sets under the restricted chase.

2. Then for the resulting rule sets we find the membership as well as data and combined complexities. These results show that without any exponential increase in the abovementioned complexities, we are able to obtain increased expressiveness for the introduced languages.

1.2 Distribution over Components for Restricted Weakly Linear Rules

Ontology-based data access (OBDA) has recently emerged as a set of technologies facilitating access to heterogeneous and incomplete data in knowledge management systems [124]. Ontologies provide a key formalism that enables effective access to such data with a unified conceptual view of various data sources and paves the way for enriching user queries with domain knowledge. Due to the presence of an ontology along with user queries in OBDA tasks, the two can be seen as components of one composite query, known as an *ontology-mediated query* (OMQ) [33].

To extract implicit knowledge from an OBDA system, reasoning is required. From a broader perspective in artificial intelligence scenarios, where even small errors are not tolerable during a decision-making process, such as a diagnosis or writing a prescription, reasoning is indispensable in providing a verifiable and easily explainable solution to interpret the results from a user query. Answering OMQs is a particular instance of reasoning in the presence of ontologies. Therefore, finding efficient ways to answer OMQs contributes to all the great capabilities that reasoning can bring in knowledge management and artificial intelligence systems.

The quest for more expressive OMQ languages has spurred major interests in existential rule languages which extend *Datalog*, a renowned query language for deductive databases. These languages enable reasoning in the presence of unknown entities. Unfortunately, answering OMQs for existential rule languages and conjunctive queries is an undecidable problem in general. To remedy this situation, many syntactic and semantic conditions were identified to make this problem decidable at the expense of limiting expressivity offered by these languages. As a result of this endeavour, fragments with various restrictions were discovered such as *MFA, linear, guarded, sticky*, and so on (cf. [56]).

A prominent application of these expressive OMQ languages is in declarative networking using which distributed computations are modeled. In these applications, queries are computed by multiple nodes over which the input database is distributed. Each node performs a local computation and the results of these computations are communicated in a master machine. A major challenge related to efficiency of these systems is provisioning of efficient reasoning.

This challenge is tackled in recent years by minimizing synchronization in computing the result of queries. In this regard, a declarative networking system is called *coordination-free* if for all input databases D there is a distribution on which the system computes $Q(D)$ without any need to do communication [8]. A central notion for provisioning coordincation-free systems is called *query distribution over components*. The question is: given an OMQ Q, whether

for every database D the answer to Q over D, denoted $Q(D)$, coincides with $\bigcup_{1\leq i\leq n} Q(D_i)$, where $D_1, \ldots, D_n$ are the (maximally connected) components of D.

The problem of checking whether an OMQ is distributable is undecidable for ontology-mediated queries based on Datalog [8]. However, for some fragments of existential rule languages, such as linear, guarded, and sticky, this problem is shown to be decidable for conjunctive queries [32].

Despite this, the scope of current decidability results is limited due to the lack of support to represent even simple constructs such as *disjunctive* axioms. In particular, disjunction, with which one can model classification, is a useful property in the biology domain among many others for which the importance of reasoning with a huge volume of the input data is most apparent.

Consider, for instance, an ontology in the domain of biological sciences which is specified by the following set of disjunctive existential rules that are studied in this chapter, which describes different types of organisms in terms of their effect on their victim. In particular, if an organism x with a weak immune system ($\text{WeakImmune}(x)$) hosts a parasitic prokaryote ($\text{Parasitic}(y)$), it gets sick by it ($\text{GetsSickBy}(x, y)$). If x gets sick by y, then y harms x ($\text{Harms}(y, x)$) and y is parasitic. The rest of the rules are self-explanatory.

$\sigma_1 : \text{Organism}(x) \rightarrow \text{Eukaryote}(x) \vee \text{Prokaryote}(x)$
$\sigma_2 : \text{Prokaryote}(x) \rightarrow \text{Bacteria}(x) \vee \text{Archaea}(x)$
$\sigma_3 : \text{Parasitic}(x) \rightarrow \exists y\, \text{Hosts}(y, x), \text{Harms}(x, y)$
$\sigma_4 : \text{Hosts}(x, y) \rightarrow \text{Organism}(x), \text{DependsOn}(y, x)$
$\sigma_5 : \text{Parasitic}(y), \text{Hosts}(x, y), \text{WeakImmune}(x) \rightarrow \text{GetsSickBy}(x, y)$
$\sigma_6 : \text{GetsSickBy}(x, y) \rightarrow \text{Parasitic}(y), \text{Harms}(y, x)$
$\sigma_7 : \text{Bacteria}(x), \text{Harms}(x, y) \rightarrow \text{Infectious}(x), \text{Victim}(y)$

In this chapter, first, we introduce a class of ontology-mediated queries constructed from what we call *restricted weakly-linear disjunctive tuple-generating dependencies* with some subsets of conjunctive queries. These rules extend a subclass of weakly-linear disjunctive Datalog introduced in [97] which we call restricted, as well as linear tuple-generating dependencies. Then we study the problem of distribution over components for ontology-mediated queries constructed from these queries and show positive results to answer this problem for some of its nontrivial fragments which we characterize in this chapter.

In particular, we provide semantic characterizations for the fragment of the above queries which distributes over connected components of the initial database. However, we show that the decidability of this problem over components for these queries cannot be established in general. Therefore we identify syntactic fragments for the aforementioned query languages for which we can establish 2ExpTime decision procedures for deciding distribution over components.

We summarize the contributions of this chapter as follows.

1. We introduce the class of restricted weakly-linear disjunctive tuple-generating dependencies which extends the classes of linear tuple-generating dependencies as well as weakly-linear disjunctive Datalog under a certain syntactic restriction.

2. We establish the first results on decidability of distribution over components for OMQs including restricted weakly-linear disjunctive tuple-generating dependencies and some subsets of conjunctive queries.

3. We empirically evaluate the distributability checking performance on real-world ontologies and compare the performance of forward chaining-based query answering for centralized and distributed scenarios on real-world ontology benchmarks.

1.3 Distribution over Components for Generalized Regular Datalog Queries

The class of *generalized regular (Datalog) queries* (GRQ) was introduced in [126]. This class of queries allows us to represent the *transitive closure* (TC) operator using which asking reachability queries can be made possible. On the other hand, adding TC to most fragments leads to undecidability of many reasoning problems in the resulting language, including equivalence of queries which is a key tool in distribution characterization. Therefore, in order to have a decidable equivalence problem, recursion is limited in this language, and it can only occur in TC rules. In fact, GRQ is shown to be the most expressive

fragment of Datalog which has a decidable equivalence problem and at the same time allows to express path queries [139].

This fragment of Datalog queries belongs to the NC class of *highly parallelizable* problems and possesses a decidable problem of query containment. In fact, recently it is identified as the solution to the long standing open problem of finding a graph query language that balances out expressiveness and tractability [126]. More recently, formal frameworks for evaluation of graph queries using regular Datalog as the query language is developed in [34, 35].

Due to forbidding recursion in GRQ queries (unless it is for expressing TC), these queries may appear to be restricted in modelling real-world databases and ontologies. However, as shown by [34, 35, 68], and as discussed by [139], we are convinced that there are a sheer number of real-world applications for which even RQs (which are GRQs restricting the maximum arity of rules to 2) are suitable modelling languages.

As an example, consider the following set $\Sigma_2 = \{\sigma_1, \sigma_2, \sigma_3\}$ of regular Datalog rules consisting of transitive predicates that describes part of a simple Bio-domain ontology, in which each predicate P^+ represents the (nonreflexive) transitive closure of P where definitions of P^+ are omitted.

$\sigma_1 : \text{Animal}(x), \text{Animal}(y), \text{Attacks}^+(x, y) \rightarrow \text{PVulnerableTo}(y, x)$
$\sigma_2 : \text{PVulnerableTo}^+(x, y), \text{StrongerThan}^+(y, x) \rightarrow \text{DistinctsEarlierThan}(x, y)$
$\sigma_3 : \text{DistinctsEarlierThan}^+(x, y) \rightarrow \text{EvolvesLongerThan}(y, x)$

The above rules describe what it means for an animal Y to be potentially vulnerable to another animal X in the animal kingdom ($\text{pVulnerableTo}(Y, X)$). Furthermore, what it means for an animal to distinct earlier than another as well as the meaning of evolving longer are specified with the second and the last rules, respectively.

In this chapter, we provide fundamental underpinnings to decide distributability over components for ontology-mediated queries constructed from fragments of generalized regular Datalog rules. Then, we show that the techniques we exploited to handle the above problem can be uniformly applied to a number of other query languages known in the literature.

We provide semantic characterizations for the fragment of the above queries

which distributes over connected components of the initial database. However, we show that the decidability of this problem over components for none of these queries can be established in general. Therefore we identify syntactic fragments for all the aforementioned query languages for which we can establish elementary decision procedures for deciding distribution over components.

We summarize the contributions of this chapter as follows.

1. The first results on decidability of distribution over components for OMQs including a syntactic fragment of generalized regular Datalog queries which we call PG2RQ are established. We then utilize the tools developed for the above OMQs in order to extend our results to a number of known query languages in the literature. Furthermore, we study the problem of distribution over components for the above query languages with the addition of transitive closure operator.

2. Then, the distributability checking performance is empirically evaluated on real-world ontologies.

Chapter 2

Introduction to Complexity Theory

Our primary purpose in this chapter is to lay theoretical backgrounds for discussing our results in this dissertation. In future chapters we will build upon this foundation to draw our results on complexity of different problems we have defined. In Section 2.1, we present the preliminaries and some basic definitions. Section 2.2 reviews the notion of polynomial hierarchy along with some of its relevant classes. In Section 2.3, some known complexity subclasses of polynomial time are introduced. Section 2.4 presents the notion of reduction and completeness. Finally, in Section 2.5, the class of ELEMENTARY is introduced. An interested reader is refered to [123] for more details.

2.1 Preliminaries

A *complexity class* is a set of functions that can be computed within a given resource. Let us now recall the complexity classes that we encounter in our complexity results. Before we proceed, let us introduce some basic concepts.

In the computational complexity analysis it is conventional to compute functions whose input and output are finite strings of bits. The set of all strings of length n is denoted by $\{0,1\}^n$, while the set of all strings is denoted by $\{0,1\}^* = \bigcup_{n\geq 0}\{0,1\}^n$. One special case of functions that map strings to strings is that of Boolean functions, for which the output is a single bit. Such a function f is defined as the set $L_f = \{x \mid f(x) = 1\}$ and we call these sets

languages or *decision problems*. The problem of computing the output of $f(x)$ given x is identified with the problem of *deciding the language* L_f which is interpreted as deciding whether $x \in L_f$, for some given x.

A decision problem P is called *decidable* or *effectively solvable* if P is a *recursive set*. Furthermore, *recursively enumerable* (RE) is the class of decision problems for which a "Yes" answer can be verified by a Turing machine in an arbitrary but finite amount of time. This means that for any RE problem there exists an algorithm that eventually halts when the answer is "Yes" but may run forever if the answer is "No". Similarly, coRE is the set of all languages that are complements of a language in RE. In other words, the membership of all languages in coRE can be disproved in a finite amount of time, however, proving membership may not halt. Recursively enumerable problems and any other problems that are not decidable are called *undecidable*.

2.1.1 Boolean Formulae and the CNF Form

A *Boolean formula* over the variables $u_1, \ldots, u_n$ consists of the variables and the logical operators AND ($\wedge$), NOT ($\neg$) and OR ($\vee$). For example, $(a \wedge b) \vee (a \wedge c) \vee (b \wedge c)$ is a Boolean formula that is $True$ iff the majority of a, b, c are $True$. If ϕ is a Boolean formula over variables $u_1, \ldots, u_n$, and $z \in \{0,1\}^n$, then $\phi(z)$ denotes the value of ϕ when the variables of ϕ are assigned the values z (1 means $True$ and 0 means $False$). A formula ϕ is *satisfiable* if there exists some assignment z such that $\phi(z)$ is True. Otherwise, $\phi(z)$ is said to be *unsatisfiable*.

A Boolean formula over variables $u_1, \ldots, u_n$ is in CNF *form (conjunctive normal form)* if it is an AND of ORs of variables or their negations. A CNF formula has the form $\bigwedge_i \big(\bigvee_j v_{ij} \big)$, where each v_{ij} is either a variable u_k or to its negation $\neg u_k$. The terms v_{ij} are called the literals of the formula and the terms $(\bigvee_j v_{ij})$ are called its clauses. A kCNF is a CNF formula in which all clauses contain at most k literals. Let SAT be the language of all satisfiable CNF formulae.

The concepts of polynomial and exponential hierarchy are the pillars of our reasoning complexities that will follow in the next chapters.

2.2 Polynomial Hierarchy

In what follows, we assume familiarity with *Turing machines* (denoted TMs); and denote the set of all decision problems solvable in polynomial time by PTIME. More formally,

Definition 1. (PTIME or P) Let $t : \mathbb{N} \mapsto \mathbb{N}$ be some function. We define DTIME$(t(n))$ to be the set of all Boolean functions that are computable in time $c \times t(n)$, for some constant $c > 0$. Then, we let: PTIME $:= \bigcup_{c \geq 0}$ DTIME(n^c).

Definition 2. (NP) A language L is in NP if for any $x \in \{0,1\}^*$, there exists a TM M and a polynomial $p(.)$:

$$x \in L \Longleftrightarrow \exists u \in \{0,1\}^{p(|x|)} \text{ s.t. } M(x,u) = 1$$

Definition 3. (coNP) The class coNP is the complementary class of NP, in which "Yes" and "No" instances are interchanged. In other words, a language L is in coNP if for any $x \in \{0,1\}^*$, there exists a TM M and a polynomial p:

$$x \notin L \Longleftrightarrow \exists u \in \{0,1\}^{p(|x|)} \text{ s.t. } M(x,u) = 1$$

or equivalently, a language L is in coNP if for any $x \in \{0,1\}^*$, there exists a TM M and a polynomial p:

$$x \in L \Longleftrightarrow \exists u \in \{0,1\}^{p(|x|)} \text{ s.t. } M(x,u) = 0$$

Definition 4. (DP) A language L is in the class DP if and only if there are two languages $L_1 \in$ NP and $L_2 \in$ coNP such that L is the intersection of L_1 and L_2.

Definition 5. (PSPACE) Let us denote by SPACE$(t(n))$, the set of all (decision) problems that are solvable by Turing machines using $\mathcal{O}(t(n))$ space for some function t of the input size n. Similarly, NSPACE$(t(n))$ denotes the set of all problems that are solvable by nondeterministic Turing machines using $\mathcal{O}(t(n))$ space. Then we define: PSPACE = $\bigcup_{k \in \mathbb{N}}$ SPACE(n^k). Note that as a corollary to Savitch's theorem, it is known that PSPACE =NPSPACE.

Definition 6. (Σ_2^{P}) A language L is in Σ_2^{P} if for any $x \in \{0,1\}^*$, there exists a TM M and two polynomials p and q:

$$x \in L \Longleftrightarrow \exists u \in \{0,1\}^{p(|x|)} \ \forall v \in \{0,1\}^{q(|x|)} \text{ s.t. } M(x,u,v) = 1$$

Note that for any language L, (i) if $L \in \mathrm{NP}$, then $L \in \Sigma_2^{\mathrm{P}}$, and (ii) if $L \in \mathrm{coNP}$, then $L \in \Sigma_2^{\mathrm{P}}$.

Definition 7. (Π_2^{P}) A language L is in Π_2^{P} if for any $x \in \{0,1\}^*$, there exists a TM M and two polynomials p and q such that:

$$x \in L \Longleftrightarrow \forall u \in \{0,1\}^{p(|x|)} \ \exists v \in \{0,1\}^{q(|x|)} \text{ s.t. } M(x,u,v) = 1$$

Definition 8. (Σ_i^{P}) For every $i \geq 1$, a language L is in Σ_i^{P} if there exists a TM M computable in polynomial-time and a polynomial $p(.)$:

$$x \in L \Longleftrightarrow \exists u_1 \forall u_2 \ldots Q_i u_i \ M(x, u_1, \ldots, u_i) = 1$$

where for all i, $u_i \in \{0,1\}^{p(|x|)}$ and $Q_i = \forall$ if i is even (resp. $Q_i = \exists$ if i is odd).

Definition 9. (Π_i^{P}) For every $i \geq 1$, a language L is in Π_i^{P} if there exists a TM M computable in polynomial-time and a polynomial p:

$$x \in L \Longleftrightarrow \forall u_1 \exists u_2 \ldots Q_i u_i \ M(x, u_1, \ldots, u_i) = 1$$

where for all i, $u_i \in \{0,1\}^{p(|x|)}$ and $Q_i = \forall$ if i is odd (resp. $Q_i = \exists$ if i is even).

Definition 10. (Polynomial Hierarchy (PH)) Based on the above-defined complexity classes, we have the following definition for the polynomial hierarchy:

$$\mathrm{PH} = \bigcup_{i \geq 0} \Sigma_i^{\mathrm{P}}$$

where $\Sigma_0^{\mathrm{P}} = \Pi_0^{\mathrm{P}} = \mathrm{P}$. One can immediately observe that: $\Sigma_1^{\mathrm{P}} = \mathrm{NP}$ and $\Pi_1^{\mathrm{P}} = \mathrm{coNP}$. Furthermore, we have $\Pi_i^{\mathrm{P}} = \mathrm{co}\Sigma_i^{\mathrm{P}} = \{\bar{L} : L \in \Sigma_i^{\mathrm{P}}\}$ and the following two inclusion properties: (i) $\Sigma_i^{\mathrm{P}} \subset \Sigma_{i+2}^{\mathrm{P}}$ and (ii) $\Sigma_i^{\mathrm{P}} \subset \Pi_{i+1}^{\mathrm{P}}$.

It is believed that $\mathrm{P} \neq \mathrm{NP}$ and $\mathrm{NP} \neq \mathrm{coNP}$. An interesting generalization of these two conjectures is that for every i, the class Σ_i^{P} is strictly contained

in $\Sigma_{i+1}^{\mathrm{P}}$. This is called the conjecture that the polynomial hierarchy does not collapse. If the polynomial hierarchy collapses this means that there is an i such that $\Sigma_i^{\mathrm{P}} = \bigcup_j \Sigma_j^{\mathrm{P}} = \mathrm{PH}$. In this case, we say that the polynomial hierarchy has collapsed to the i-th level. The smaller i, the weaker, and hence more plausible, the conjecture that PH does not collapse to the i-th level.

In the following, we show two collapsing results in PH.

1. If PTIME=NP, then PH=PTIME *i.e.*, the hierarchy collapses to PTIME.
2. If $\Sigma_i^{\mathrm{P}} = \Pi_i^{\mathrm{P}}$ for some $i \geq 1$, then for any $j \geq 1$, $\Sigma_j^{\mathrm{P}} = \Pi_j^{\mathrm{P}} = \Sigma_i^{\mathrm{P}}$.

2.2.1 Oracle Characterization of Polynomial Hierarchy

We can alternatively, define PH with oracle machines. Before we proceed, let us define oracle Turing machines.

Oracle machines are machines that are given access to an oracle which can *magically* solve the decision problem for some language $L \subseteq \{0,1\}^*$. This machine has a special oracle tape on which it can write a string $l \in \{0,1\}^*$ in one step, to get an answer to the following question: *Is l in L?*. Notice that we can repeat this arbitrarily often with different queries. In complexity theory, a *difficult language* is the one which cannot be decided in polynomial time. If a language L is of this kind, then this oracle provides an additional power to the TM.

Definition 11. (Oracle Turing Machines) An oracle Turing machine is a TM M that possesses a special read/write tape, which is called M's oracle tape, and three special states q_{query}, q_{yes}, q_{no}. To execute M, in addition to the input, a language $L \subseteq \{0,1\}^*$ is specified that is used as the oracle for M. During the execution of this TM, whenever M enters the state q_{query}, if $q \in O$, then the Turing machine moves into the state q_{yes} and if $q \notin O$, then it moves into q_{no} in which q denotes the contents of the special oracle tape. No matter what the choice of O is, a query that asks the membership in O counts only as a single computational step. If M is an oracle machine, $O \subseteq \{0,1\}^*$ is a language, and $x \in \{0,1\}^*$, then the output of M on input x and with oracle O is denoted by $M^O(x)$. We define nondeterministic oracle TMs similarly.

Definition 12. For every $O \subseteq \{0,1\}^*$, PTIME^O is the set of languages decided by a polynomial-time deterministic TM with oracle access to O and NP^O is the set of languages decided by a polynomial-time nondeterministic TM with oracle access to O.

Definition 13. Let ϕ be a Boolean formula and for all $i \geq 1$, u_i be a vector of Boolean variables, and $Q_i = \exists$ if i is an odd number and $Q_i = \forall$, otherwise. We define the problem $\Sigma_i\mathsf{SAT}$ as follows:

$$\exists u_1 \forall u_2 \exists u_3 \ldots Q_i u_i \phi(u_1, \ldots, u_i) = 1$$

Recall that we execute oracle machines given access to queries of the following form: "is $q \in O$ for some language O?". For every $O \subseteq \{0,1\}^*$, an oracle machine M and input x, by $M^O(x)$ we denote the output of M on x with access to oracle O. We have the following characterization of the polynomial hierarchy:

Proposition 14. *For every $i \geq 2$, Σ_i^{P} is defined to be $\text{NP}^{\Sigma_{i-1}\mathsf{SAT}}$ which is the class of languages decided by polynomial time nondeterministic Turing machines with access to oracle $\Sigma_{i-1}\mathsf{SAT}$.*

Clearly, if we have oracle access to the hardest problems of a class $\mathcal{C}$, we can solve all problems in $\mathcal{C}$. Therefore, in some texts, the class names are used in the oracle notation. For example, $\text{NP}^{\Sigma_1\mathsf{SAT}}$ is denoted NP^{NP}, *etc.* We use the latter convention in our notations of this dissertation.

The following properties are known to hold for the above-defined classes:

$$\Sigma_0^{\text{P}} = \Pi_0^{\text{P}} = \text{PTIME} \tag{2.2.1}$$

$$\Sigma_i^{\text{P}} \subseteq \Sigma_{i+1}^{\text{P}} \tag{2.2.2}$$

$$\Pi_i^{\text{P}} \subseteq \Pi_{i+1}^{\text{P}} \tag{2.2.3}$$

$$\Sigma_i^{\text{P}} = co\Pi_i^{\text{P}} \tag{2.2.4}$$

2.3 Sub-polynomial Classes of Complexity

In this section, we briefly review the sub-polynomial classes of complexity: AC and NC which are the classes of efficiently parallelizable problems, and

LogSpace and NLogSpace which are the deterministic and nondeterministic classes of problems that are efficiently solvable with low resources on space. In addition to their theoretical significance, these complexity classes play a key role in real-world applications due to the efficiency benefits they offer for parallelizability of computational tasks to solve problems that belong to these classes. As we will see in Chapter 7 of our dissertation, the high level of parallelizability offered by these classes can help us establish practical benefits for query answering using the query languages introduced therein which have the above complexities of reasoning.

2.3.1 Circuit Complexity Model

Circuit complexity is an area with a long history which starts in the 1940's. Loosely speaking, it is a branch of computational complexity theory in which we classify Boolean functions according to the size or the depth of the Boolean circuits that compute them.

An *n-bit Boolean function* is a Boolean function that has domain $\{0,1\}^n$ and co-domain $\{0,1\}$. The basic question circuit complexity is trying to answer is: given a collection of "simple functions" $\mathcal{F}$ and a target Boolean function g, how efficiently can one compute g on all inputs using $\mathcal{F}$? We usually measure efficiency using the "size" of computation which asks how many copies of $\mathcal{F}$ are necessary to compute g?

Let F be a set of Boolean functions, which in this setting we call a *basis set*. The fan-in of some $f \in F$ is the number of inputs that f takes. Unbounded fan-in means that f can take any number of inputs. We define a *Boolean circuit* C with n inputs $x_1, \ldots, x_n$ and size s over a basis $\mathcal{F}$ as a directed acyclic graph which consists of n sources (aka inputs $x_1, \ldots, x_n$) and one sink or output node (which is the sth node in some fixed topological order on the nodes). The nodes are also called *gates*. Input gates are $x_1, \ldots, x_n$ in which x_i is associated with the ith input bit. Additionally, we label the gates numbered $j = n+1, \ldots, s$ in a fixed topological order with a function $f_j \in \mathcal{F}$.

The size of C is the number of AND and OR gates it contains. The depth of C is the longest path from any source to the sink. The fan-in of some $f \in \mathcal{F}$

is the number of inputs that f takes. The most popular basis only uses AND, OR, and NOT gates, where NOT gates are not counted toward size.

Definition 15. (AC) Let AC^i for some integer $i \geq 0$ denote the languages recognized by Boolean circuits of depth $\mathcal{O}(\log^i n)$ and a polynomial number of unlimited fan-in AND and OR gates. Then, we define $\mathrm{AC} = \bigcup_{i \geq 0} \mathrm{AC}^i$.

As a particular subclass of the AC hierarchy, AC^0 is appealing as it corresponds to first-order logic in the model checking setting. More precisely, the evaluation of the first-order logic (i.e., SQL) queries over relational databases, with only database as the input, belongs to AC^0. Intuitively, a problem belongs to AC^0 if it can be decided in constant time using a polynomial number of processors in the size of the input.

We will provide backgrounds on the first-order logic and relational databases in more details in Chapter 3.

Definition 16. (NC) Let us define NC^i as the fan-in 2 version of AC^i. Then, analogous to the AC^i hierarchy, we define $\mathrm{NC} = \bigcup_{i \geq 0} \mathrm{NC}^i$.

Equivalently, a problem P is in NC if there exist a pair of constants c and k such that P can be solved in time $\mathcal{O}(\log^c n)$ using $\mathcal{O}(n^k)$ parallel processors. It is not hard to see that NC is a subset of PTime. The reason is that we can always simulate polylogarithmic parallel computations by polynomial-time sequential ones. However, it is highly suspected to be false, it is still unknown whether NC = PTime. If the above equality can be established, it implies that there are tractable problems that are inherently sequential and cannot be therefore significantly sped up using parallelism.

Definition 17. (LogSpace and NLogSpace) The set of all problems that are solvable by deterministic (resp. nondeterministic) Turing machines using a logarithmic amount of space is called LogSpace (resp. NLogSpace). Formally, LogSpace = Space$(\log n)$ and NLogSpace = NSpace$(\log n)$.

The following inclusion relationships are known from the literature for the sub-polynomial complexity classes:

$$\mathrm{NC}^0 \subseteq \mathrm{AC}^0 \subset \mathrm{NC}^1 \subseteq \textsc{LogSpace} \subseteq \textsc{NLogSpace} \subseteq \mathrm{AC}^1 \subseteq \cdots \subseteq \mathrm{NC} = \mathrm{AC} \subseteq \textsc{PTime}$$

In particular, it is known that AC^0 is strictly contained in LogSpace, while it is open whether the other inclusions are strict. The problem of *undirected graph reachability* is an example of a problem that belongs to LogSpace but not in AC^0 [125].

2.4 Reducibility and Completeness

In theory of computational complexity, a reduction is a transformation of one problem into another. Depending on the transformation used, this can be used to define complexity classes on a set of problems.

Roughly speaking, if a problem A reduces to a problem B, this means that A is no harder than B, and equivalently B is no easier than A. $A \leq B$ is written, usually with a subscript on $\leq$, to indicate the type of reduction being used. More formally we have:

Definition 18. Given two subsets A and B of $\mathbb{N}$ and a set of functions F from $\mathbb{N}$ to $\mathbb{N}$ which is closed under composition, A is called reducible to B under F if:

$$\exists f \in F \ \forall x \in \mathbb{N} \ \ x \in A \Leftrightarrow f(x) \in B$$

which is written as $A \leq_F B$.

Definition 19. Let S be a subset of $P(\mathbb{N})$ and let $\leq$ be a reduction. S is called *closed under* $\leq$ if:

$$\forall s \in S \ \forall A \in P(\mathbb{N}) \ \ A \leq s \Rightarrow A \in S$$

A subset A of $\mathbb{N}$ is called *hard* for S if $\forall s \in S \ s \leq A$. In addition, a subset A of $\mathbb{N}$ is called *complete* for S if A is hard for S and A is in S.

2.4.1 Reductions

The two main types of reductions that are used in the literature of computational complexity are *many-one* and *Turing* reductions. Many-one reductions map instances of one problem into instances of another. On the other hand, Turing reductions compute the solution to one problem, assuming the other

problem is easy to solve. Many-one reductions are weaker than Turing reductions.

A problem P is said to be *complete* for a complexity class $\mathcal{C}$ if every problem in $\mathcal{C}$ reduces to P, and P itself is also in $\mathcal{C}$. As in this case any solution to the problem, in combination with the reductions, can be used to solve each and every problem in the class, the problem in this sense represents the class.

To show that a decision problem P is undecidable we need to find a reduction from a decision problem, already known to be undecidable, to P. Note that the reduction function must be computable (i.e., there must be some Turing machine which computes the reduction in finite time). In the following, we introduce the most common reductions in the literature.

Turing Reduction

In computational complexity theory, a Turing reduction from a problem A to a problem B is intuitively, a reduction which can easily solve B, given the assumption that A is easy to solve. More formally, a Turing reduction is a function which is computable by an oracle machine using an oracle for A. In this case, if it is shown that such a reduction exists, then every algorithm to solve M immediately yields an algorithm which solves L. This solution is formed by inserting a *call* to that algorithm at each place which is used by the oracle machine.

Notice that since there is a possibility that the oracle machine invokes the algorithm many times, the resulting algorithm may asymptotically require more time to be solved than either the oracle machine or M. Further, it could require as much space as both algorithms combined. It is known that Turing reductions can be applied to both decision as well as function problems.

It is known that many important complexity classes, such as NP are not closed under Turing reductions. In particular, any decision problem can be Turing-reduced to its complement. This can be done by simply solving the original problem and inverting the answer. Using this fact, we can show that any complexity class which is not closed under its complement is also not closed under Turing reductions. However, it can be shown that some classes within

PTIME (such as LOGSPACE, NLOGSPACE, and PTIME itself) are closed under Turing reductions.

There are indeed situations where Turing reductions are too powerful. In this case, we need to utilize a special case of these reductions instead. One well known special case of these reductions is known in the literature as many-one reductions. In fact it can be shown that most natural complexity classes are closed under the latter class of reductions. Informally speaking, many-one reductions can be seen as Turing reductions where the oracle can be invoked only once at the end of the processing of the oracle machine.

Often times, additional resource restrictions are taken into account for Turing reductions. As instances of these restrictions one can say that the oracle machine runs in polynomial time or logarithmic space; see polynomial-time reduction and logspace reduction which follow, for details.

Many-One Reduction

In computational complexity theory, a many-one reduction is a reduction which converts instances of a decision problem A into instances of a decision problem B. To denote that there is a many-one reduction from A to B, we write $A \leq_m B$, or in other words A *is many-one reducible to* B. If an algorithm N solves instances of a problem B, we can exploit N to solve instances of A in the time (or the space) that is needed to run N plus the time needed to apply the reduction.

More formally, suppose A and B are formal languages over the alphabets Σ_1 and Σ_2, respectively. A many-one reduction from A to B is a total computable function $f : \Sigma_1^* \to \Sigma_2^*$ that has the following property: If such a function f exists, then it is said that A *is many-one reducible to* B.

Further, a class $\mathcal{C}$ of languages is said to be closed under many-one reduction if there exists no reduction from some language in $\mathcal{C}$ to a language outside of $\mathcal{C}$. If a class is closed under many-one reduction, then many-one reduction can be used to show that a problem is in $\mathcal{C}$ by reducing a problem in $\mathcal{C}$ to it.

Many-one reductions are important classes of reductions as most well studied complexity classes are indeed closed under some type of many-one reduc-

tion. These classes include, but not limited to, PTime, NP, LogSpace, NLogSpace, coNP, PSpace, ExpTime, etc. Recall that many-one reductions are a special case and a weaker form of Turing reductions as in the former only one invocation of the oracle is allowed, which happens only at the end.

It turns out that in many cases Turing reductions are more convenient for designing reduction algorithms. However, their power also forbids several important classes such as NP to be closed under these reductions. On the other hand, in many cases many-one reductions are subjected to additional resource restrictions. As instance of these restrictions one can consider the function to be computable in polynomial time or logarithmic space; see polynomial-time reduction and logspace reduction for details.

Polynomial-Time Reduction

In computational complexity theory, a *polynomial-time reduction* is a reduction which is computable by a deterministic Turing machine which runs in polynomial time. Depending on the type of reduction used, it is called differently. If it is a many-one reduction, then it is called a *polynomial-time many-one reduction*, *polynomial transformation*, or *Karp reduction*. Unless otherwise stated, in this dissertation, any reduction that is mentioned or proved is a Karp reduction. To denote the fact that there is a polynomial-time reduction from instances of a decision problem A to instances of a decision problem B, we usually write $A \leq_p B$. If it is a Turing reduction, then it is called a *polynomial-time Turing reduction*, which is also known as *Cook reduction*.

Be it a Cook or a Karp reduction, polynomial-time reductions are significant and widely-used, as that they offer a balance: they are powerful enough to perform many transformations between important problems, however, they are weak enough in a sense that polynomial-time reductions from problems in NP or coNP to problems in PTime are considered unlikely to exist. This is the notion of reducibility that is used in the standard definitions of several complete complexity classes, including, but not limited to, NP-complete, PSpace-complete, and ExpTime-complete.

Note that indeed, it is inappropriate to consider and apply polynomial-

time reductions within the class PTime. The reason is that any problem in PTime can be polynomial-time reduced (for both many-one and Turing variants) to any other problem in PTime. Thus, for classes within PTime such as LogSpace, NLogSpace, and PTime itself, we use logspace reductions instead. It is known that if a problem has a Karp reduction to a problem in NP, the problem is in NP. Cook reductions are known to be more powerful than Karp reductions. For instance, any problem in coNP has a Cook reduction to a problem in NP. While it is useful to exploit this power to design reductions, there is one downside too: classes such as NP are not closed under Cook reductions, and so they are not useful for proving that a problem is in NP. However, they are useful for showing that problems are in PTime.

Logspace Reduction

In computational complexity theory, a logspace reduction is a reduction which is computable by a deterministic Turing machine using logarithmic space. In a conceptual level, this means that these reductions can keep a constant number of points into the input, along with a logarithmic number of integers that are of a fixed size. Notice that for such a machine there are polynomially-many configurations, and for this purpose, logspace reductions are also trivially polynomial-time reductions. It is widely conjectured that logspace reductions are weaker than polynomial-time reductions; while any language in PTime is polynomially reducible to any other language in PTime, a logspace reduction between a language in NLogSpace and a language in LogSpace, which are both subsets of PTime, would imply the equality: LogSpace = NLogSpace which indeed, seems unlikely to hold. It is an open question whether NP-complete problems behave differently under logspace and polynomial-time reductions. Often times, logspace reductions are used on languages inside the class of PTime, in which case it usually does not matter whether many-one or Turing type of these reductions are used. The reason is that it is known that LogSpace, NLogSpace, and PTime are all closed under Turing reductions which means that Turing reductions can be used to show that a problem belongs to any of these languages. However, there are

other subclasses of PTIME which are not known to be closed under Turing reductions, and therefore for these subclasses many-one reductions need to be used instead. As much as polynomial-time reductions are useless within PTIME and its subclasses, logspace reductions are useless to distinguish problems in LOGSPACE as well as its subclasses. In particular, every problem in LOGSPACE is trivially LOGSPACE-complete under logspace reductions. While even weaker reductions are known to exist, they are not often used in practice, because languages smaller than LOGSPACE receive little attention in the literature.

Note that the complexity classes PTIME, NP and PSPACE are closed under polynomial-time reductions. Furthermore, it is known from the literature that the complexity classes LOGSPACE, NLOGSPACE, PTIME, NP and PSPACE are closed under logspace reduction.

2.4.2 Complete Problems in Polynomial Hierarchy

Based on Definition 19, we have the following definition for completeness of different levels of polynomial hierarchy.

Definition 20. For all $i \geq 0$, a language L is Σ_i^{P}-complete if $L \in \Sigma_i^{\mathrm{P}}$ and for all $L' \in \Sigma_i^{\mathrm{P}}$, $L' \leq_p L$. Π_i^{P}-completeness is defined similarly.

Proposition 21. *If there is a language L that is* PH*-complete, then there exists some $i \geq 0$ such that* PH $= \Sigma_i^{\mathrm{P}}$*,* i.e.*, the hierarchy collapses to its i-th level.*

2.5 ELEMENTARY

The complexity class EXPTIME is the class of all problems that are solvable by a deterministic Turing machine in $\mathcal{O}(2^{p(n)})$ time, where $p(n)$ is a polynomial function of input n. Similarly, the complexity class NEXPTIME is the set of problems that can be solved by a nondeterministic Turing machine using time $2^{n^{\mathcal{O}(1)}}$. Formally, NEXPTIME $= \bigcup_{k \in \mathbb{N}} \mathrm{NTIME}(2^{n^k})$, where $\mathrm{NTIME}(t(n))$ is the class of all problems that are solvable by nondeterministic Turing machine in time $c \times t(n)$, for some constant c.

More generally, for all $i \geq 1$, we define:

$$i\,\textsc{ExpTime} = \bigcup_{k \in \mathbb{N}} \textsc{DTime}(2^{2^{\cdot^{\cdot^{2^{n^k}}}}})$$

where the number of exponentials in the above equation is i. Similarly, for all $i \geq 1$,

$$\textsc{N}\,i\,\textsc{ExpTime} = \bigcup_{k \in \mathbb{N}} \textsc{NTime}(2^{2^{\cdot^{\cdot^{2^{n^k}}}}})$$

Now, we let the complexity class ELEMENTARY of elementary recursive functions be the union of i ExpTime classes as follows:

$$\text{ELEMENTARY} := \textsc{ExpTime} \cup \textsc{2ExpTime} \cup \textsc{3ExpTime} \cup \cdots$$

Furthermore, co$\textsc{N}\,i\,\textsc{ExpTime}$ is the complementary class of $\textsc{N}\,i\,\textsc{ExpTime}$, in which "Yes" and "No" instances are interchanged.

In summary, the following inclusion relationships are known between the discussed complexity classes:

$$\text{NP}, \text{coNP} \subset \Sigma_2^{\text{P}}, \Pi_2^{\text{P}} \subset \textsc{PSpace} \subset \textsc{ExpTime} \subset \textsc{NExpTime},$$

$$\textsc{coNExpTime} \subset \textsc{2ExpTime} \subset \textsc{N2ExpTime}, \textsc{coN2ExpTime}$$

Chapter 3

Background, Motivations, and Previous Developments

In this chapter we go over state of the art in this area and review some of the work related to this dissertation. Later, in the next chapters we will build on top of the state of the art in our development. In Section 3.1, we lay out the landscape of ontology-mediated query answering along with its comparison with other formalisms. Section 3.2 reviews some of the related work along with applications of the languages introduced in this dissertation. Finally, in Section 3.3, we discuss the previous work on chase termination.

The informed readers can read this chapter selectively, or skim it in the first reading.

3.1 The Landscape of Ontology-Mediated Query Answering

Since the seminal work of [124], there has been a surge of growing interest both in academia and from industry in ontology-mediated query answering (OMQA). It is a new paradigm in data management the purpose of which is to exploit the semantic knowledge provided by an ontology to improve query answering. Ontology is a way to express the semantic connections between different pieces of data which is expressive enough to capture complicated relationships in the data while at the same time mathematically formalized and restrictive enough to allow software as well as humans to make inferences

and provide reasoning.

There are certain advantages in adding ontologies which are summarized below:

- It provides an enriched vocabulary that matches closely the conceptualization of users in the given application domain. This way, an ontology facilitates query formulation of users.

- In addition, the ontology can be used to integrate different data sources through a single conceptual model. This makes it easy for them to be accessed in a uniform and intuitive manner.

- Furthermore, OMQA can help to support automated reasoning with which one can uncover implicit connections between terms; and also it aids in detecting modelling errors.

- Last but not least, OMQA can provide users with more complete answers to their queries. This can mean unveiling hidden connections in the data which is not known using any other way. OMQA can provide this by taking into account not only the facts explicitly stored in the data, but also facts that are implicit consequences in the data as well as the domain knowledge.

For the sake of illustration, let us consider an example of an OMQA scenario in medicine. The patient data of hospitals is usually recorded in databases. For this purpose, a standard medical ontology is used as a terms repository.

One may represent the domain knowledge that a patient suffers from *acute lymphoblastic leukemia* and has been prescribed a drug for hypertension. If an administrator queries the system to find all cancer patients that are being treated for high-blood pressure, in the absence of an ontology, this query may show no results. The reason is that in many real-world cases, the generic terms 'cancer' and 'high blood pressure' do not appear in the records of patients, explicitly.

However, the ontology defines a hierarchy of terms from specialized to generic ones and clarifies the semantic relationships between them and in or-

der to answer a query, an OMQA system will perform the necessary inferences given the knowledge base (including ontology and fact base). In this example, it can infer that acute lymphoblastic leukemia is a type of cancer and hypertension is the same as high blood pressure.

Note that OMQA is not limited to medical data and can be broadly applied to scientific data (*e.g.*, chemistry, biology), bibliographical data, governmental open data, *etc*.

In spite of all advantages that are provided by OMQA, enriching data with domain knowledge has its cons. In particular, it makes the task of query answering significantly more difficult, both conceptually and algorithmically. This also depends on which language is used to express the ontological knowledge.

Fortunately, there are known classes of ontologies for which the task of query answering is easy. However, to achieve this goal, the syntax of the language that describes the ontology needs to be severely restricted and this may lead to a considerable loss of expressiveness in the domain ontology.

Another challenge that arises when enriching data with ontologies is the increasing volume of data. This increases the complexity of query answering and renders this task infeasible in reality even for very simple queries. In this dissertation we provide solutions for both of the above mentioned challenges.

In what follows, we further elaborate on ontologies as compared to traditional database systems. Using our settings, a knowledge base is defined as a pair consisting of a database and an ontology where ontologies model some aspects of a particular domain of interest.

Ontologies do this by introducing the vocabulary that is relevant to the particular domain of interest we are modelling. This vocabulary typically includes *classes*, *relations* and *hierarchies*. In the above example, classes are: cancer, acute lymphoblastic luekemia, hypertension, high blood pressure. Furthermore, relations are: same-as and is-a. In addition, hierarchies are: Acute lymphoblastic luekemia $\leq$ Cancer.

On top of domain vocabulary, ontologies formally specify rich semantics and meaning to the vocabulary. These semantics are typically represented in

terms of rules. For the above example, this translates to the following sentences in plain English: "Every Acute lymphoblastic luekemia is-a Cancer"; "Hypertension is the same-as High blood pressure".

3.1.1 Ontologies versus Database Schemata

In this section, we compare an ontology and database schema briefly.

As for similarities, they both describe the structure and constraints on data. However, they are different in the following aspects: (i) While ontologies entail implicit facts, database schemata define legal database states; (ii) Although ontologies may severely influence query answering (QA), database schemata do not influence QA. In addition, in many database applications, a closed world assumption (CWA) is made, which means that the information that is missing from the database is assumed to be false. For ontologies, however, open world assumption (OWA) is made in which the missing information is unknown (*i.e.*, neither assumed to be true nor false).

The ontology languages that are frequently used in OMQA are description logics and existential rules languages. In what follows, we demonstrate a logical view of database related to the above example.

Example 1. *Consider the following tables that belong to a hospital database.*

Table 3.1: A hospital database

Is-Diagnosed-with

Ada	Cancer
Linus	Cancer
Ada	High Blood Pressure
Richard	High Blood Pressure
Andy	z_0

Patient

Ada
Linus
Alice
Richard
Bob

One can represent the above tables using the language of first-order logic as follows:

$$\exists z_0\big(\textit{is-diagnosed-with}(Ada, Cancer) \wedge \textit{patient}(Ada) \wedge$$
$$\textit{is-diagnosed-with}(Linus, Cancer) \wedge \textit{patient}(Linus) \wedge$$
$$\textit{is-diagnosed-with}(Ada, \textit{High-Blood-Pressure}) \wedge \textit{patient}(Alice) \wedge$$

$$is\text{-}diagnosed\text{-}with(Richard, High\text{-}Blood\text{-}Pressure) \wedge patient(Richard) \wedge$$
$$is\text{-}diagnosed\text{-}with(Andy, z_0) \wedge patient(Bob)\big)$$

For example, consider the following relational algebra operations on the above tables.

1. $\pi_{[1]}(Is\text{-}Diagnosed\text{-}with) \bowtie Patient$
2. $\pi_{\varnothing}(\sigma_{[1=Andy]} Is\text{-}Diagnosed\text{-}with)$

These queries will be translated to FO logic queries as shown below in the same order.

1. $Q() = \exists x \exists y\big(is\text{-}Diagnosed\text{-}with(x, y) \wedge Patient(x)\big)$
2. $Q() = \exists x\big(Is\text{-}Diagnosed\text{-}with(Andy, x)\big)$

More formally, it was shown by Codd that every relational algebra expression can be translated to one in FO logic [2].

Theorem 22. *Relational algebra is equivalent to FO without function symbols.*

It can also be shown that answering queries in this setting is seen as finding homomorphisms from the query atoms to set of facts in the database as is demonstrated in the following example.

Example 2. *Consider the queries of Example 1. Let*

$$Q_1 = \exists x \exists y\big(is\text{-}Diagnosed\text{-}with(x, y) \wedge Patient(x)\big)$$

Then answering Q_1 is equivalent to finding all variable substitutions h from the variables of Q_1 to the constants of the database such that $h(Q_1) \subseteq F$, where F is the set of database facts. This is equivalently denoted as: $F \vDash Q_1$. For Q_1, we can find the following substitutions:

$$h_1 = \{x \mapsto Ada, y \mapsto Cancer\}$$
$$h_2 = \{x \mapsto Linus, y \mapsto Cancer\}$$
$$h_3 = \{x \mapsto Ada, y \mapsto HighBloodPressure\}$$
$$h_4 = \{x \mapsto Richard, y \mapsto HighBloodPressure\}$$

The same argument holds for $Q_2 = \exists x\big(Is\text{-}Diagnosed\text{-}with(Bob, x)\big)$ *and we find the following substitution for* Q_2*:* $h_5 = \{z_0\}$*. Therefore, we can compute the following answers to* Q_1 *and* Q_2*:* $Answer(Q_1, F) = \{(Ada), (Linus), (Richard)\}$ *and* $Answer(Q_2, F) =$ "Yes".

Conjunctive queries form the sub-language of relational algebra which are obtained using the only following operations: projection, Cartesian product, and selection with equality. Q_1 and Q_2 in Example 1 are two examples of conjunctive queries.

These languages may be further extended, to get a more expressive sub-language of *union of conjunctive queries*, by adding union operation to them. Unions of conjunctive queries are therefore defined as unions of a finite number of conjunctive queries.

Note that relational algebra and relational calculus have substantial expressive power. In particular, they can express natural join, unions of conjunctive queries, *etc*. However, they cannot express recursive queries (in which roughly, the definition of the query, directly or indirectly, depends on itself).

Beyond First-Order Logic: Transitive Closure

The *transitive closure* (TC) of a binary relation R is the smallest (binary) relation S for which $R \subseteq S$ and S is transitive. As a particular example of recursion, it was shown by Fraisse, that there does not exist any expression in relational algebra that defines the TC of a given binary relation E (for more information cf. e.g., [2]). This means that one cannot write an FO formula using predicate symbols R and T that will be satisfied in any model if and only if T is the TC of R.

In order to express TC of a relation R one needs an infinite formula:

$$\{(x,y)\,|R(x,y) \vee \exists z(R(x,z) \wedge R(z,y)) \vee \exists z,v(R(x,z) \wedge R(z,v) \wedge R(v,y)) \vee \exists z,v,w(R(x,z) \wedge R(z,v) \wedge R(v,w) \wedge R(w,y)) \vee \dots\}$$

The intuition behind inability of relational calculus to express TC is that it can only express *"local"* properties. Locality informally means that in order to check if a tuple belongs to the result of a query, only a certain predetermined portion of the input need to be looked at.

To overcome the limitations of relational calculus in terms of expressive power, two approaches can be considered. The first is to embed relational calculus commands inside a conventional programming language. The other approach is to augment relational calculus with a high-level mechanism for recursion to keep declarative nature of this language. Due to keeping the declarative nature of relational calculus the latter approach is a better solution.

Datalog language was an answer to this shortcoming of relational algebra which augments the language of conjunctive queries with a recursion mechanism. This language gained traction in the past few years due to many applications even outside database community including networking, access control strategies, business analytics, *etc*. It is a truly declarative language and is syntactically a subset of *Prolog*. Note that it is not Turing complete as it belongs to the function free fragment of Prolog.

The languages that have been considered for expressing ontologies in the literature are in two main categories of *description logics* (DLs) and *positive existential rules*. For the former, the most prominent subcategories are of DL-Lite, EL and OWL2 profiles. The latter family of languages that are the focus of this dissertation are known as tuple-generating dependencies (TGDs) (also known as Datalog$^{\exists}$) that generalize Horn DLs.

The main notion central to this research, the chase, is an algorithmic tool for constructing universal models that repeatedly applies a series of steps in order to accomplish certain tasks, such as reasoning under the presence of ontologies, repairing inconsistent database instances, *etc*. Utilizing the chase algorithm for OMQA is also known as *saturation* or *materialization* approach. Each chase step takes a TGD (aka a *dependency*) that is not satisfied by the database instance, a set of tuples (relational instances) that violate the dependency and changes the database instance so that the resulting instance satisfies the dependency for the given set of tuples.

As an example, consider a database instance D containing tuples $P(a, b)$ and $Q(a, a)$ and a dependency specified by the following rule

$$\forall x \forall y P(x, y) \rightarrow Q(y, x)$$

which intuitively says that for each tuple $P(x, y)$ there needs to be a corresponding tuple of the form $Q(y, x)$. The given dependency is not satisfied for the tuple $P(a, b)$, because there does not exist a corresponding tuple $Q(b, a)$ in D as specified by the dependency. In this case, the chase step will simply add the missing tuple $Q(b, a)$ to D and the resulting instance will be $D' = \{P(a, b), Q(a, a), Q(b, a)\}$. It is easy to see that the resulting instance satisfies the dependency for the tuple $P(a, b)$, and that it actually satisfies the dependency for all sets of tuples in D'.

The problem becomes more challenging when existential quantification is allowed in the head of a rule, e.g.,

$$\forall x Person(x) \rightarrow \exists y Friend(x, y)$$
$$\forall x \forall y Friend(x, y) \rightarrow Person(y)$$

which says every person has a friend and a friend is a person. Given a database with some individual persons, chase is nonterminating, as with an instance of x, $\exists y Friend(x, y)$ does not tell us any concrete individual that can serve as a substitute for existential variable y, which can thus only be an unknown individual.

The chase procedure has its origins in two seminal papers of 1979, one by Aho *et al.* [6] and the other by Maier *et al.* [111]. Maier *et al.* used chase as a tool to check if a set of embedded dependencies Σ, specified by a set of *functional dependencies (FDs)* and *join dependencies (JDs)*, logically implies a given join dependency ξ. For this, they represented ξ as a tableau chased with Σ and checked if the resulting tableau represents the identity mapping for all instances that satisfy Σ. Also, the chase was reformulated for other types of dependencies [140].

In [24], a unified treatment was proposed for the implication problem by extending the chase procedure for classes of tuple-generating and equality-generating dependencies. These two classes of dependencies were shown to be expressive enough to capture all the previous classes.

More recently, the chase procedure has gained a lot of attention due to its usefulness in: data integration [44, 108], data modelling and ontologies [42, 43], inconsistent databases and data repairs [4, 10, 86], data exchange [66],

peer data exchange [69], and data correspondence [86].

Since the evaluation of chase might not terminate and it is undecidable whether it terminates for any given database and also for all databases [1] [74, 87], the problem of defining (decidable) sufficient conditions ensuring termination has received a great deal of interest in recent years. In this regard, several termination criteria have been proposed. One of the main weaknesses of current approaches is that they are limited in scope as they mostly focus on Skolem chase and there is no systematic analysis of termination under the restricted variant of the chase.

Inspired by applications of chase algorithm in the aforementioned database problems, the goal of the first technical part of this dissertation is to tackle the hard problem of finding conditions for chase termination for all database instances under the *standard chase variant*, which is in general strictly more powerful than Skolem chase. This leads to devising systematic algorithms for discovering classes of tuple generating dependencies with finite universal models beyond finite Skolem chase.

3.2 Related Work

In this section, we review some of the most recent literature that are relevant to our research.

3.2.1 Ontology Modelling

Ontologies are fundamental and key concepts in the semantic web. Also they are proved to be useful in databases, because of their flexibility and expressive power, most prominently in data modelling and data integration. Among ontology formalisms, description logics (DLs) have been playing a key role in research in semantic web much of which directed toward scalable and efficient query answering over ontologies. In particular, the DLs of the DL-Lite family

[1] Note that in case of any given database, database is part of the input but when we say for all databases, database is not given in input. In the above cited papers, different undecidability results have been derived for these cases for standard and core variants of the chase.

[48] are the most common DLs in the semantic web and databases that allow for tractable query answering.

In the next section, we introduce another ontology modelling formalism introduced recently which is known to be more expressive than most DLs for ontological modelling.

Datalog$^{\pm}$

Datalog$^{\pm}$ is a rule-based formalism, introduced by Cali, *et al.* in [43], that combines the advantages of *logic programming (LP)* in Datalog with features for expressing ontological knowledge like adding key modelling features including existential quantification in rule heads (TGDs), constraints of the form: $body \rightarrow X = Y$ (equality generating dependencies or EGDs) and constraints of the form: $body \rightarrow \bot$. The sign " + " in Datalog$^{\pm}$ is due to additionally allowing existentially quantified variables in the head of rules (for increasing expressiveness), and the sign " − " is due to restrictions enforced on rule bodies to guarantee decidability and tractability properties. First, we say a few words about the old (plain) Datalog.

Datalog is a recursive database query language defined in the 1980s [70]. It provides a simple but useful framework for inductive definitions. Datalog can be considered as function-free Prolog with fully declarative semantics. Plain Datalog serves as a basis for knowledge representation (KR) languages (such as the popular OWL 2 RL profile of the web ontology language) and also is seen as a common subsumer for a variety of very expressive query languages (cf. [36, 37]).

Moreover, Datalog is quite often used as a target for knowledge compilation from much more expressive KR languages for instance, a recent topic of research has been to show how tractable description logics or classes of existential rules can be reformulated using Datalog for query answering purposes [56, 81, 97, 122].

In the past few years Datalog has gained a renewed interest due to its successful usage in recent applications in web data extraction, code querying, modelling and automation and in fact many large projects and some companies

are "Datalog-based" (*e.g.*, [54, 62, 127]);

Datalog is composed of a finite number of rules of the following form:

$$\langle condition_1 \rangle, \langle condition_2 \rangle, \ldots, \langle condition_N \rangle \rightarrow \langle result \rangle$$

where the right hand side is an atom, called the *head* of the rule, and the left hand side is a conjunction of atoms called the *body* of the rule. An atom is a predicate, or relation name with arguments. Also, extensional database predicates (or EDBs) are considered source tables and intensional database predicates (or IDBs) derived tables. As an example of Datalog rules, consider the following where IDB is the set of atoms derivable from the given rule set and the given EDB.

Figure 3.2.1 shows a Datalog program where the EDB includes a number of facts about employees' names (represented by *emp*/1 predicates) and a list of reports from/to (represented by *reports*/2 predicates) and the IDB includes derived facts from the given EDB and rule set.

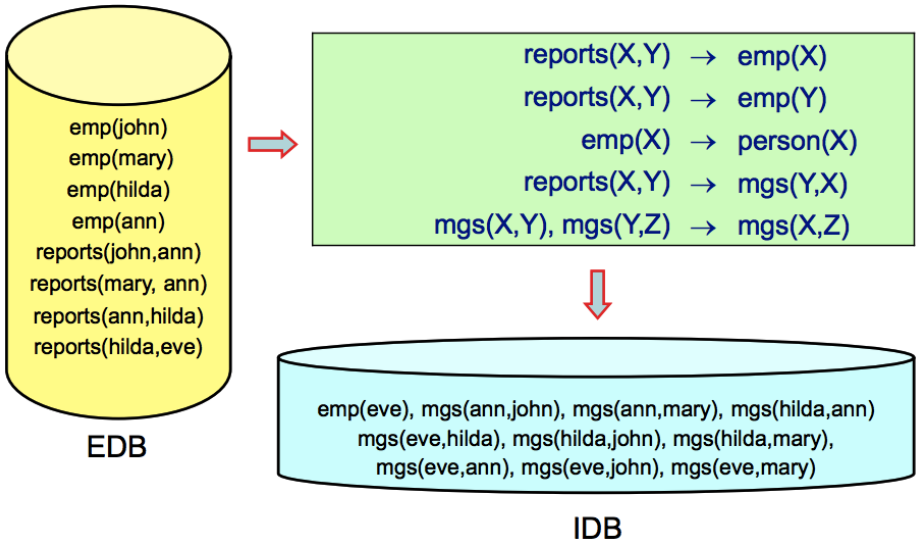

Figure 3.2.1: Datalog example

Datalog has recently been subject to different theoretical analyses such as [128, 129] where a complete picture for expressiveness of Datalog and its extensions by input negation (semi-positive Datalog) and/or a linear order (order-invariant Datalog) have been considered.

There are many applications of Datalog$^{\pm}$ family of languages among which are data exchange, data extraction, ontology-based data access, ontology querying, querying the semantic web, etc. In the following, we explain data exchange.

3.2.2 Data Exchange

Data exchange is an old database problem that only recently earned more formal treatment. More precisely, it is the problem of transforming data structured under a source schema to data structured under a different target schema.

The leading project for data exchange and data integration was *Clio project* [64], developed at the IBM Almaden research center since 1999 aiming for a working system for schema-mapping generation and data exchange.

Clio pioneered the use of schema mappings, specifications that describe the relationship between data in two heterogeneous schemata. From this high-level, nonprocedural representation, it can automatically encode, to transform data from one representation to another for data exchange or for generating either a view or to reformulate queries against one schema into queries on another for data integration purposes.

Formally, a schema mapping is defined as a triple $M = (S, T, \Sigma)$, where S is a source schema and T is a target scheme and Σ is composed of high-level, declarative assertions to specify the relationship between S-instances and T-instances.

Ideally, schema mappings must be expressive enough to specify data interoperability tasks and also simple enough to be efficiently manipulated by tools.

A natural specification language for schema mappings is the language of first-order logic. However, since the problem of satisfiability of first-order logic sentences is undecidable in general, using this language, checking solutions of schema mapping is undecidable as well. So, we need to restrict ourselves to *well behaved* fragments of first-order logic as our specification language. Each schema-mapping specification language must support the following properties:

Copying: Each source table must be able to be copied to a target table and renamed:

$$\forall x_1, ..., x_n \; P(x_1, ..., x_n) \rightarrow Q(x_1, ..., x_n)$$

Projection: A target table can be formed by deleting one or more columns of a source table:

$$\forall x_1, x_2, x_3 \; P(x_1, x_2, x_3) \rightarrow Q(x_1, x_2)$$

Column Addition: A target table can be formed by adding one or more columns to a source table:

$$\forall x_1, x_2 \; P(x_1, x_2) \rightarrow \exists z Q(x_1, x_2, z)$$

Decomposition: A source table can be decomposed into two or more target tables:

$$\forall x_1, x_2, x_3 \; P(x_1, x_2, x_3) \rightarrow Q_1(x_1, x_2) \wedge Q_2(x_2, x_3)$$

Join: A target table can be formed by joining two or more source tables:

$$\forall x_1, x_2, x_3 \; P_1(x_1, x_3) \wedge P_2(x_3, x_2) \rightarrow Q(x_1, x_3, x_2)$$

All of these features can be found in TGDs as a specification language and in fact, they can be specified using a special class of TGDs known as source-to-target TGDs (s-t TGDs) which are in the following form: $\forall \mathbf{x} \phi(\mathbf{x}) \rightarrow \exists \mathbf{y} \psi(\mathbf{x}, \mathbf{y})$, where $\phi(\mathbf{x})$ (resp. $\psi(\mathbf{x}, \mathbf{y})$) is a conjunction of atoms over the source (resp. target) schema. In addition to source-to-target dependencies, we also consider target dependencies which are defined in the same format of s-t TGDs with the difference that the body and the head are both over target schema.

Data exchange via the schema mapping $M = (S, T, \Sigma)$ is to transform a given source instance I to a target instance J, such that (I, J) satisfies the specifications Σ of M.

More formally, data exchange setting is a quadruple $(S, T, \Sigma_{st}, \Sigma_t)$, where S represents the source schema, T represents the target schema, Σ_{st} is a set of constraints representing the relationship between the source and target

schema, and Σ_t represents a set of constraints over the target schema. Given a data exchange setting $(S, T, \Sigma_{st}, \Sigma_t)$ and the instance I over the source schema S, the data exchange problem is to find instances J over the target schema T, such that $I \cup J$ is a model for D and $\Sigma_{st} \cup \Sigma_t$. An instance J with the previous properties is called a solution to the data exchange problem, or simply a solution. This problem was first formalized by Fagin *et al.* in [65].

Most of the data exchange problems consider Σ_{st} to be a set of TGDs and Σ_t to be a set of TGDs and EGDs. The data exchange setting we consider here is of this format.

Figure 3.2.2 demonstrates a graphical representation of data exchange problem in which S and T represent source and target schemata, I and J are the source and target instance, Σ_{st} is the set of source-target TGDs and Σ_t is the set of target TGDs.

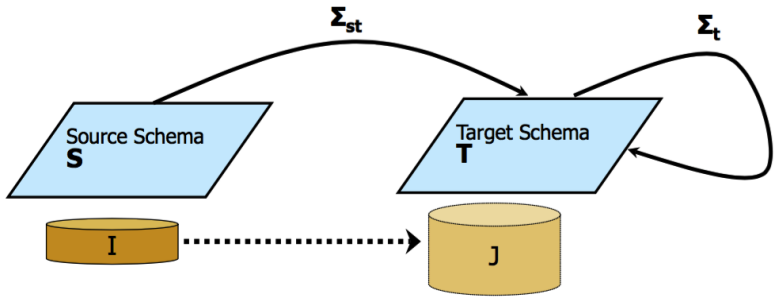

Figure 3.2.2: Data exchange setting

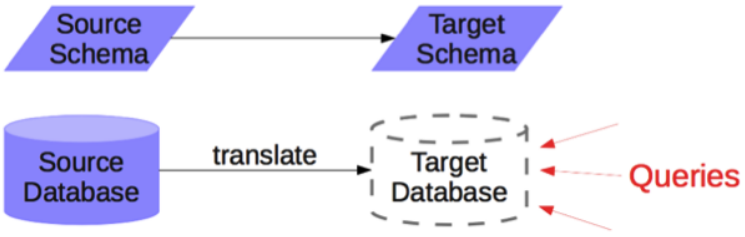

Figure 3.2.3: Answering queries in data exchange

Note that given a source instance, there may exist multiple solutions. For example, considering the following data exchange setting specified by

$(S, T, \Sigma_{st}, \Sigma_t)$ where:

$$S = \{P\}, T = \{Q\}, \Sigma_{st} = \{P(x, y) \rightarrow \exists z Q(x, z) \wedge Q(z, y)\}, \Sigma_t = \varnothing$$

and the source instance is $I = \{P(a, b)\}$, there are countably infinitely many solutions as shown below:

$J_1 = \{Q(a, b), Q(b, b)\}$

$J_2 = \{Q(a, a), Q(a, b)\}$

$J_3 = \{Q(a, n_1), Q(n_1, b)\}$

$J_4 = \{Q(a, n_1), Q(n_1, b), Q(a, n_2), Q(n_2, b)\}$, etc.

Where $n_1, n_2, ...$ are labeled nulls and a and b are constants. So, now the question is: which solutions are *better* than others? and how can we compute the *best* answer? Answer to this question defines the semantics of data exchange.

This question was answered by Fagin *et al.* in [65] by introducing a *universal solution* which is a solution that has homomorphisms to all other solutions (thus, being the *most general* solution). Roughly, a homomorphism $h : J_1 \rightarrow J_2$ between two target instances is a structure-preserving mapping from domain of J_1 to domain of J_2 that maps constants to themselves and furthermore, if $R(a_1, \ldots, a_n)$ is in J_1, then $R(h(a_1), \ldots, h(a_n))$ is in J_2. Looking at the previous example, we find out that J_1 and J_2 are not universal but J_3 and J_4 are universal solutions. Note that if J_1 and J_2 are universal solutions for I, then they are homomorphically equivalent, which is not the case.

Based on their notion, Fagin *et al.* defined universal solutions as the *preferred solutions* in data exchange.

As an example, consider the following data exchange setting: $(S, T, \Sigma_{st}, \Sigma_t)$ where:

$S = \{Employee\}, T = \{EmpDept\}, \Sigma_{st} = \{Employee(x) \rightarrow \exists z EmpDept(x, z)\},$

$\Sigma_t = \varnothing$ and the source instance is: $I = \{Employee(mia), Employee(liz)\}$.

For this example, the following instances over target schema represent solutions:

$J_1 = \{EmpDept(mia, dept_1), EmpDept(liz, dept_1)\}$

$J_2 = \{EmpDept(mia, dept_1), EmpDept(liz, dept_2)\}$

$J_3 = \{EmpDept(mia, dept_1), EmpDept(liz, dept_1), EmpDept(ben, dept_1),$
$EmpDept(john, dept_3)\}$

$J_4 = \{EmpDept(mia, dept_3), EmpDept(sam, dept_1), EmpDept(liz, dept_2),$
$EmpDept(sally, dept_1)\}$

On the other hand, we notice that the instance $J_5 = \{EmpDept(mia, dept_1)\}$ is not a solution since the information that liz must be given a department is not reflected in J_5. For this example, we find out that if z is replaced with any null values (say z_1 and z_2 which can be a placeholder for department names), we can come up with instance J_6 on the target schema which is indeed a solution for the given source instance I: $J_6 = \{EmpDept(mia, z_1), EmpDept(liz, z_2)\}$.

Based on the above informal definitions, we find out that this solution is universal since for any solution K, we can construct instance J_K by replacing nulls z_1 and z_2 in J_6 with some constants or nulls such that $J_K \subseteq K$.

In the following, we introduce more recent applications of (some fragments of) the languages we introduce in this dissertation. Note that the technical development of this work does not depend on these issues and the reader can safely skip Sections 3.2.3 and 3.2.4.

3.2.3 Semantic Big Data Warehousing

Combining data from various sources let us create a more informative picture of a domain of interest. Data integration is the way to achieve this goal. A

data warehouse is an example of data integration for analytical purposes. It is constructed by integrating data from multiple heterogeneous sources that support analytical reporting, query answering services, and decision making.

The evolution of data storage techniques and architectures, swayed the direction of research in data warehousing and integration as well.

Unfortunately, semantic data integration, particularly for scientific data, is a very challenging topic. The first challenge is accessing the data in the first place. Data can be in different forms and in different locations which include files on a computer, or in tables within PDF documents. In addition, these data may be inconsistent. Moreover, there may be access restrictions as some of the repositories may be accessible via websites or structured query mechanisms while others may need usage of secure file modifications (*e.g.*, edit, copy or transfer). Last but not least, Some data might be stored in proprietary databases or require special and expensive software to access the materials. *Linked Open Data* was an endeavor to address these issues.

As an example, consider a scenario in the medical domain. During a visit to a doctor, a patient is assigned a primary diagnostic of "Acute upper respiratory infection" using ICD-10[2], which is the 10^{th} revision of the global health information standard for mortality and morbidity statistics.

Now, assume the lab test results of that patient be: "Virus is identified in Nose by Culture" which is encoded using LOINC, a database and universal standard for identifying medical laboratory observations. Given the above information, how can we understand relevance of the above result to the diagnosis of the patient?

This is where semantic data integration techniques fits in to help in complex analysis and querying.

Ontologies have been proposed as a way to express the semantic connections between different pieces of data in a way that is both machine and human readable. However, the challenges still exist when ontologies are taken into account, in particular when the datasets are large and complex. Moreover, big data introduces challenges which complicates data warehousing. A compre-

[2]https://www.who.int/classifications/icd/factsheet/en/

hensive list of challenges of the semantic data integration can be found in [53].

3.2.4 Knowledge Graphs

Following the trend of large technological giants like Google, LinkedIn, Amazon, and Facebook, it is becoming common for enterprises to integrate their heterogeneous sources of information into a unified structure known as a *knowledge graph*. A knowledge graph typically consists of graph-structured data to allow for seamless accommodation of changes in the structure of the data that is used for the validation and also enrichment of data and knowledge extraction from data by uncovering of hidden insights from it. NELL [117], DBpedia [107], YAGO [135] and Wikidata [143] are some of the most well known knowledge graphs.

Different models and standards of graph data structures are deployed in industrial-grade applications. For example, triple stores utilize RDF[3] which are deployed in *e.g.*, GraphDB[4], AllegroGraph[5], AnzoGraph[6], Apache Jena[7], *etc.*

In particular, GraphDB has clients like AstraZeneca and the well known news agency BBC, with a focus on publishing data and data integration scenarios, where RDF shines the most. These corporations demand 500 million to 1 billion RDF facts for their purposes. Furthermore, the biggest cluster installations go up to 15 billion RDF facts. On the other hand, all structured knowledge in *e.g.*, Wikipedia is less than 800 million facts. Although the problem of scalability has been mostly addressed in successful triple stores, but it certainly remains a challenge when querying in the presence of ontologies.

Property graphs are another form of graph structured data. They are used

[3]https://www.w3.org/RDF
[4]http://graphdb.ontotext.com/
[5]http://www.franz.com/agraph/allegrograph
[6]http://www.cambridgesemantics.com/product/anzograph/
[7]http://jena.apache.org

by graph database vendors such as Neo4j[8], OrientDB[9], Amazon Neptune[10], *etc*.

As opposed to (RDF) triple stores (or quad stores), that are edge-centric, graph databases are node, or property, centric. On the other hand, triple-store can handle trillions of records and support inferencing on data, making them great to provisioning analytics. They utilize URIs (uniform resource identifiers), which provide support for querying and reasoning about the Semantic Web. This is in contrast with graph databases which do not support inferencing. Note that inferencing is supported by ontologies.

As an example for comparison of graph databases and RDF stores, let us consider a knowledge representation scenario in which we want to represent the fact that a person named "Alice" knows a person named "Bob". In RDF, it is demonstrated as:

```
<http://example.org/person/1> :hasName "Alice".
<http://example.org/person/1> foaf:knows <http://example.org/
person/2>.
<http://example.org/person/2> :hasName "Bob".
```

while in the graph database Neo4j it is written as:

```
(a:Person {name: "Alice"})-[:KNOWS]->(b:Person {name: "Bob"})
```

Graph-structured data may also derive from relational or semi-structured data which exhibits graph structure. Each knowledge graph comes equipped with a query language. In particular, Neo4j uses Cypher[11], SPARQL[12] is used to query RDF datasets, *etc*. This enriched knowledge graph can then be the input of any data science toolkit for graph data processing and computing which may apply statistical and machine learning techniques on it to provide graph analytics.

[8]http://https://neo4j.com/
[9]https://orientdb.com/graph-database/
[10]http://aws.amazon.com/neptune/
[11]https://neo4j.com/developer/cypher-query-language/
[12]https://www.w3.org/TR/rdf-sparql-query/

New ontology languages might develop into suitable formalism for expressing views and logical relationships over knowledge graphs, but providing the required integration with other artificial intelligence approaches is in its infancy.

MARPL is a prominent ontology language recently introduced for knowledge graphs [114]. MARPL theories are in what is known as multi-attributed predicate logic (MAPL) with the ExpTime-complete data complexity. In [144], it is shown that the data complexity of the Skolem chase is PTime-complete. Therefore, MARPL theories cannot be encoded in any Skolem chase terminating rule set. On the other hand, in [104] it is shown that the restricted chase allows encoding of problems with data complexities higher than PTime and even nonelementary. It is conjectured that there is a translation from MARPL theories to existential rules and based on what was discussed above, they require restricted chase reasoning and they cannot be encoded in any Skolem chase terminating rule set. Note that as of March 2020, MARPL offers no support for integration with other AI techniques [103].

The Vadalog system is another prominent ontology language for knowledge graphs [26]. It is a Datalog-based reasoning system which utilizes Warded Datalog$^{\pm}$ as a decidable fragment of Datalog$^{\pm}$. This language captures Datalog as well as SPARQL queries under OWL 2 QL entailment regime and it allows integration of big data processing techniques into one unified system.

The Vadalog ontology language has a PTime data complexity, while allowing ontological reasoning. *DeepReason*[13] is a recent industrial spinout that provides explainable reasoning services on knowledge graphs based on Vadalog.

3.2.5 Disjunctive Reasoning

Disjunctive rules extend Datalog by allowing uncertainty in reasoning in situations where the exact reasoning is not possible. However, a closed set of alternatives are given for each predicate. This knowledge can be represented using disjunction operator. For example, any person can be identified as a male or a female. This knowledge can be represented using the following disjunctive

[13]http://deepreason.ai/

Datalog rule.

$$person(x) \rightarrow male(x) \vee female(x)$$

This is useful when modelling classification tasks. Since this construct is a very useful one, a flurry of activity has been done toward understanding semantics and complexity of rules that include disjunction in the consequent of rules. Disjunctive Datalog rules were first introduced in [63]. Complexity of ontological reasoning under disjunctive existential rules has been addressed in [82].

3.2.6 Reasoning at Scale

More recently, the issue of scalability of OMQA (aka *rule-based reasoning*) has been brought into attention due to the large volume of data produced as a bi-product of big data movement.

The current approaches for rule-based scalable reasoning based on materialization are divided into the two categories of *centralized* and *distributed materialization*. The former provides scalability of reasoning on a single machine using in-memory architectures. The latter, on the other hand, deploys multiple machines for provisioning scalability.

Distributed materialization is further divided into *inter-rule* and *intra-rule* distribution architectures. In the former, the input data is replicated on each machine and each rule is executed concurrently. In the latter, the data is partitioned and the full set of rules are executed only on the data that belongs to that particular partition (which is locally available). Each one of these approaches has its own pros and cons for different situations. In general, developing technologies that enable in-memory computing and parallel processing is highly challenging, even for current database architecture systems [94].

There has been a lot of effort for scaling reasoning tasks in the realm of the Semantic Web. For instance, WebPIE [138] is a MapReduce-based reasoner based on intra-rule approach.

Except for a few recent works regarding in-memory reasoning engines which

handle existential rules [30, 120, 137], we are not aware of any practical attempt for scalable reasoning using existential rule languages.

OMQA Based on DLs

Recently there has been a surge of interest in effective reasoning on scale in the semantic web. For reasoning with large ABoxes when DLs are used as the underlying ontology language, the abstraction refinement approach has been introduced in [72]. The goal of this method is to speed up reasoning using an abstract small ABox. This approach is shown to be sound and complete for Horn $\mathcal{ALCHOI}$ [73] as well as Horn $\mathcal{SHOIF}$ DLs. Even though this approach is shown to be sound for more expressive ontologies such as $\mathcal{SROIQ}$, it is not complete for such ontologies. This will help speed up the materialization computation for less expressive ontologies. DLs restrict the maximum arity size of rules to be 2. Therefore, the expressiveness is limited when this language is considered for the purpose of OMQA.

OMQA Based on Existential Rules

Distributed reasoning involving existential rules is extremely challenging due to introduction of fresh nulls (aka Skolem terms) during the process of reasoning which may lead to the nontermination of corresponding computation. In the literature, many sufficient conditions have characterized existential rules for which the computation of chase termination or checking if there is an answer to a given query is decidable (cf. [88]). These conditions impose sometimes strict syntactic restrictions on rules. Moreover, the complexity of checking some of these conditions can go up to 2ExpTime-complete or more.

Chase engines compute the set of derived atoms (*i.e.*, conclusions of a given existential rule set and a given database). The state of the art chase engines such as RDFox [120], Graal [17], VLog [137], PEGASUS [116], PDQ [30] have been optimized to handle large databases in different ways. They apply centralized materialization through the use of shared-memory architectures in order to provide scalability.

For example, RDFox utilizes a multicore architecture to avoid any locking

mechanism that prevents parallelism. For this purpose, RDFox utilizes hash tables to store the database and the derived facts. VLog, on the other hand, uses a number of optimizations including deploying columnar layout rather than conventional row-by-row architectures for data storage which makes it easy for implementation of compression techniques such as Run Length Encoding (RLE), *etc.*

As an instance of such optimizations, consider the following set of relations are stored in the database: $\{R(a_1, b_1), R(a_1, b_1), R(a_1, b_2)\}$. The columnar layout would store these relations using two columns: $c_1 = \langle a_1, a_1, a_1 \rangle$ and $c_2 = \langle b_1, b_1, b_2 \rangle$. After performing RLE on c_1 and c_2, we have: $c_1' = \langle a_1, 3 \rangle$ and $c_2' = \langle b_1, 2, b_2 \rangle$, respectively. Note that the number which follows a constant in the database represents the number of occurrences of that constant in the relation that corresponds with the column. An interested reader can find a comprehensive comparison analysis on different chase engines and a number of benchmarks in [31], which is the first benchmark on the chase to date.

On the other hand, currently, the study on distributed materialization for OMQA based on existential rules is lacking in the literature. The theoretical problem of distributed query answering using existential rules has been considered for the first time in [32]. It has been shown that the decision problem of checking whether a set of existential rules is distributable is undecidable in general.

However, distribution over components can be decided for guarded and sticky existential rules is decidable and the underlying problems are ELEMENTARY and coNEXPTIME-complete, respectively. So, this approach, although theoretically possible, seems to be of a very high complexity to be applicable in practice.

Moreover, on the practical side, no previous work on reasoning with existential rule ontologies has evaluated distributed reasoning focusing on existential rule sets.

On the second part of this dissertation, we consider the problem of distributed OMQA and develop theoretical and practical techniques for OMQAs based on some fragments of disjunctive TGDs.

3.3 Previous Work on the Chase Termination

Since our technical development is often related to, or compared with, the state of the art, let us introduce some key classes of the finite chase here and comment on the latest developments related to our work conducted in Chapters 4 and 5. Note that all acyclicity conditions that are given below ensure the termination of the Skolem chase, and therefore, of the restricted chase, except for RMFA and RJA which ensure the termination of the restricted chase and allow to identify more terminating rule sets.

We introduce the following notations: Given a rule set R, for each predicate P which occurs in the schema of R (the set of predicates appearing in R), let us denote the ith argument of P with $P[i]$. For all predicates P occurring in a given rule set R and all integers $i \in \{1, \ldots, arity(P)\}$ (where $arity(P)$ is the number of arguments of P), let us call each $P[i]$ a position. Furthermore, for each variable x which occurs in R, let $pos_B(x)$ (resp. $pos_H(x)$) denote the set of body (resp. head) positions in which x occurs. Every time a rule which consists of at least one existential variable in a given rule set is applied, it generates what is known as a new Skolem term or a fresh null in each application. Let $r \in R$ be a rule in a given rule set R. We denote the set of all existentially (resp. universally) quantified variables of r with $var_{ex}(r)$ (resp. $var_u(r)$). The set of all existentially (resp. universally) quantified variables of a given rule set R, denoted $var_{ex}(R)$ (resp. $var_u(R)$) is the union of $var_{ex}(r)$ (resp. $var_u(r)$) for all rules $r \in R$.

3.3.1 Terminating Classes of Existential Rules

In this part, we introduce some of the key classes of terminating chase introduced in the literature. The discussion here is technical in nature.

Weakly-acyclic (WA) [66], roughly speaking, tracks the propagation of terms in different positions. A rule set is *weakly acyclic* (*WA*) if there is no position in which Skolem terms including Skolem functions can be propagated cyclically, possibly through other positions.

More formally, given a rule set R, the (position) dependency graph WA(R)

for R contains positions as vertices. Moreover, for each existential rule $r \in R$, each universal variable x which occurs in R, each position $P[i] \in pos_B(x)$, and each existential variable y which occurs in R, WA(R) has a normal edge from $P[i]$ to each $Q[j] \in pos_H(x)$ and a special edge from $P[i]$ to each $Q[j] \in pos_H(y)$. A rule set R is WA if WA(R) does not have a cycle which involves a special edge. Consider the following rule set $R_1 = \{r\}$

$$r : P(x, y) \rightarrow \exists z\, P(y, z)$$

It is easy to verify that WA(R_1) involves a special edge from $P[2]$ to itself. Indeed, R_1 may lead to the construction of an infinite P-chain of new elements (*i.e.*, Skolem terms).

Now, consider the following rule set $R_2 = \{r\}$, where $r : B(y), P(x, y) \rightarrow \exists z\, P(y, z)$. The dependency graph of R_2 contains the same cycle as before. However, R_2 cannot be applied recursively. The reason is that invented terms (fresh nulls or new Skolem terms) do not need to belong to B.

In fact, weakly acyclicity overestimates the chase termination as it may introduce fake cycles which may not cause infiniteness of the chase procedure for the given rule set like the above example demonstrates. To rule out such fake cycles, more general conditions have been introduced.

Joint-acyclic (JA) [105] generalizes WA as follows. Let R be a rule set. For each variable $y \in var_{ex}(R)$, let $Move(y)$ be the smallest set of positions such that

(i) $pos_H(y) \subseteq Move(y)$; and

(ii) for each rule $r \in R$ that $var_{ex}(r) \neq \varnothing$ and for all variables $x \in var_u(r)$, if $pos_B(x) \subseteq Move(y)$, then $pos_H(x) \subseteq Move(y)$.

The *JA dependency graph* JA(R) of R is defined as: the set of vertices of JA(R) is $var_{ex}(R)$, and there is an edge from y_1 to y_2 whenever the rule that contains y_2 also contains a variable $x \in var_u(R)$ such that $pos_H(x) \neq \varnothing$ and $pos_B(x) \subseteq Move(y_1)$. $R \in$ JA if JA(R) does not have a cycle. It is easy to verify that R_2 belongs to JA.

Acyclic graph of rule dependencies (aGRD) A rule set R belongs to the *acyclic graph of rule dependencies* (aGRD) class of acyclic rules if there is no cyclic dependency relation between any two (not necessarily different) rules of R, possibly through other dependent rules of R. To define the rule dependency graph [12, 18] of a rule set R, we introduce the rule dependency relation $\prec \subseteq R \times R$ as follows. Consider two rules $r_1, r_2 \in R$ such that $r_1 = body(r_1) \rightarrow \exists \mathbf{z_1}\ head(r_1)$ and $r_2 = body(r_2) \rightarrow \exists \mathbf{z_2}\ head(r_2)$. Let $sk(r_1) = body(r_1) \rightarrow sk(head(r_1))$ and $sk(r_2) = body(r_2) \rightarrow sk(head(r_2))$. Then, $r_1 \prec r_2$ if and only if there exists an instance I, substitutions θ_1 (resp. θ_2), for all variables in $sk(r_1)$ (resp. $sk(r_2)$) such that $\theta_1(body(r_1)) \subseteq I$, $\theta_2(body(r_2)) \subseteq I \cup \theta_1(sk(head(r_1)))$, and $\theta_2(body(r_2)) \notin I$. R has an acyclic graph of rule dependencies if $\prec$ on R is acyclic. In this case, R is called aGRD.

Note that the original definition of aGRD in [12] considers fresh nulls as opposed to Skolem terms, which based on [88] does not change the resulting relation $\prec$.

Model-faithful acyclic (MFA) [56] is a semantic acyclicity class of the Skolem chase which generalizes all the Skolem acyclicity classes mentioned above. A rule set R is MFA if in the Skolem chase of R w.r.t. the critical database of R (i.e., the database which contains all possible ground atoms based on predicates of R and the single constant symbol $*$ without any occurrence in R), there is no cyclic Skolem term (a term with at least two occurrences of some Skolem function).

Restricted joint acyclicity (RJA) [49] (originally defined for disjunctive existential rules) generalizes JA and is defined using what is known as the restricted dependency graph $\mathcal{G}$ of a rule set R. To the best of our knowledge, this condition was the first notion for restricted chase termination defined in the literature. Intuitively, similar to JA, RJA is based on checking acyclicity of a graph constructed from the existentially quantified variables of the given rule set R. However, in addition to checking the condition as required by JA to evaluate whether there is an edge from a node to another, RJA checks another condition called nonblocking, which intuitively evaluates if the head of a rule is satisfiable or not.

More formally, consider a graph $\mathcal{G}$, called the restricted graph of a given rule set R, constructed from R in which the set of nodes of $\mathcal{G}$ are the existentially quantified variables of R. Let $\mathcal{R}_{\mathsf{dng}}(\mathcal{F})$ denote the set of facts obtained from a given set of facts $\mathcal{F}$ by exhaustive application of all rules of R without existential quantifiers. Let further, B_n and H_n denote body and the head of the rule $n \in \{n_1, n_2\}$ occurs in and μ the substitution that replaces all variables z by fresh constants $\langle z, * \rangle$. There is an edge from n_1 to n_2 if n_2 occurs in an existential rule $r \in R$ with a frontier variable $x \in \mathbf{x}$, such that the following conditions hold: (i) all body positions of x occur in n_1; and (ii) for the set $\mathcal{F} = \mu(B_{n_2} \cup H_{n_1}[n_1/x] \cup B_{n_1})$, we have $\mathcal{R}_{\mathsf{dng}}(\mathcal{F}) \not\models \mu(H_{n_2})$. R is called *restricted jointly acyclic (RJA)*, if the restricted dependency graph of R does not have any cycle.

Restricted model-faithful acyclicity (RMFA) [49] generalizes MFA as follows. Let R be a non-disjunctive rule set. For each rule $r \in R$ and each homomorphism h such that h is a homomorphism on $body(r)$, $C_{h,r}$ is defined as the union of $h(body(r))$, where each occurrence of a constant is renamed so that no constant occurs more than once, and F_t for each Skolem term t in $h(body(r))$, where F_t is the set of ground atoms involved in the derivation of atoms containing t. Let RMFA(R) be the least set of ground atoms such that it contains the critical database of R and let $r \in R$ be a rule and h a homomorphism from $body(r)$ to RMFA(R). Let $v \in var_{ex}(r)$ be some existential variable of r. If $\exists v.h(head(r))$ is not logically entailed by the exhaustive application of non-generating (Datalog) rules on the set of atoms $C_{h,r}$, then $h(sk(head(r))) \subseteq$ RMFA(R). We define $R \in$ RMFA if RMFA(R) contains no cyclic Skolem terms.

In this dissertation we show the set of all terminating rule sets under some condition $\mathcal{C}$ with $\mathcal{C}$. Then, the following inclusions are known from the literature: WA $\subset$ JA, JA $\subset$ MFA, aGRD $\subset$ MFA, JA $\subset$ RJA, MFA $\subset$ RMFA. Furthermore, WA and aGRD are not comparable.

3.3.2 Other Results Related to Chase Termination

In [49], a notion known as *restricted model-faithful cyclicity* (RMFC) has been introduced which provides a sufficient condition for deciding nontermination of the restricted chase of a given rule set for all databases. Intuitively, RMFC is based on detecting cyclic functional terms in the result of the exhaustive application of *unblockable* rules on the grounded version of $body(r) \cup sk(head(r))$ for some generating rule r, such that in the mapping used for the grounding, each variable x is replaced by some fresh constant c_x.

To characterize a sufficient condition of termination of a given rule set for arbitrary databases, for any chase variant, it would be useful to have a special database that can serve as a *witness* for proving termination. Let us refer to it as a *critical database* I^*. Having such a critical database in place guarantees that given a rule set R, if there is some database that witnesses the existence of an infinite chase derivation of R, then I^* is already such a witness database. If we know that such a critical database exists for some chase variant, then we can focus on sufficient conditions to decide the chase termination of a rule set w.r.t. I^*.

From [113], it is known that such a critical database exists for the oblivious and Skolem chase variants. The construction of such a critical database for those chase variants is also easy: Let R be a rule set. Let C denote the set of constants appearing in R and let $*$ be a special constant with no occurrence in R. A database is a (Skolem) critical database if each relation in it is a full relation on the domain $C \cup \{*\}$. With this measure in place, it is then easy to show why all the known classes of terminating rule sets under Skolem and oblivious chase variants (such as the aforementioned acyclicity conditions) work well. The reason is that they rely on this critical database.

However, for the restricted chase, no critical database exists. Note that for the terminating conditions of RJA and RMFA [49] that are the only known concrete criteria for the termination of restricted chase rules, the introduced "critical databases" are ad hoc in that they do not provide a *principled way* to construct such a database that may lead to more general classes of terminating

rule sets under the restricted chase. In fact, due to the nature of the problem, which is not recursively enumerable [85], as also pointed out in [75], finding such a critical database even for subsets of rules with syntactic (or semantic) restrictions is very challenging. More recently, termination of linear rules under the restricted chase has been considered in [106], where the body and the head of rules are composed of singleton atoms (called *single-body* and *single-head* rules). As part of this work, the existence of such a critical database is proved by simply showing a database consisting of a single atom.

Also, for single-head *guarded* and *sticky* rules, the same problem has been considered in [75], where the authors characterize nontermination of restricted chase sequences constructed from the aforementioned rule sets using sophisticated objects known as *chaseable sets* which are infinite in size. For this purpose, they show that the existence of an infinite chaseable set characterizes the existence of an infinite restricted chase derivation. In particular, for guarded rule sets, the latter can be strengthened with the fact that we can focus on acyclic databases to show the decidability of restricted chase termination for guarded TGDs.

Furthermore, for sticky TGDs, this is shown via the existence of a *finitary caterpillar* which is an infinite *path-like* restricted chase derivation of some database the existence of which can be checked via a deterministic *Büchi automaton*. Their work is focused only on single-head rules and, to the best of our knowledge, no characterization exists for multi-head rules. This is unlike the Skolem chase for which the notion of δ-bounded ontologies have been defined uniformly using the (Skolem) critical database technique [144].

The decision problem of termination of the oblivious and the Skolem chase variants have been considered for linear and guarded rules in [40], and this problem is shown to be PSpace-complete and 2ExpTime-complete, respectively, for linear and guarded rules. More recently, the same problem has been considered for sticky rules in [41], and it has been shown to be PSpace-complete. This shows that for these rules, sufficient and necessary conditions can be established to decide termination.

It is worth mentioning that similar to our work, in [15], a tool was intro-

duced to extend different (Skolem) acyclicity conditions ensuring chase termination. However, unlike our approach, their extension never extends a Skolem chase terminating rule set to a terminating one under the restricted chase.

Also related to the work of Chapters 4 and 5, the notion of *k-bounded rules* was introduced in [60] for oblivious, Skolem, and restricted chase variants. The k-boundedness problem they considered in that work checks whether, independently from any database, there is a fixed upper bound of size k on the number of breadth-first chase steps for a given rule set, where k is an integer. For arbitrary values of k, this problem is already known to be undecidable for Datalog rules (TGDs without existential variables, also known as range-restricted TGDs [2]), as established in [92] and [112].

The breadth-first chase procedure in [60] refers to chase sequences in which rule applications are prioritized. Their prioritization is in a way that those rule applications which correspond to a particular breadth-first level occur before those that correspond to a higher breadth-first level. Under the assumption that k is excluded from the input, and only the rule set is given as the input, they prove an ExpTime upper bound for checking k-boundedness for the oblivious and the Skolem chase variants and 2ExpTime upper bound for the restricted chase.[14]

Notice that as discussed in [60], TGDs with k-boundedness property are *union of conjunctive queries-rewritable* (or *UCQ-rewritable*, also known to belong to *finite unification sets* of TGDs (or *fus*) [18]). It is worth mentioning that this latter work has a different scope from ours in that, unlike k-bounded TGDs of [60], the k-$\mathsf{safe}(\Phi_\Delta)$ rule sets that is our contribution in this chapter, where Δ is some Skolem acyclicity condition, already generalize Datalog rule sets (for any value of $k \geq 0$), and therefore, are not UCQ-rewritable. Besides, there is no characterization of any critical database for the restricted chase variant in [60] which is a key issue and the focus of the current chapter and then one that follows.

[14]Note, however, that if k is part of the input, i.e., when the problem is: *given a rule set R and a unary-encoded integer k, whether R is k-bounded for the considered chase*, the complexity of the problem is in 2ExpTime and 3ExpTime for the aforementioned chase variants, respectively.

Acyclicity Condition	Membership checking complexity	Data complexity	Combined complexity
WA	PTime	PTime-*c*	2ExpTime-*c*
JA	PTime-*c*	PTime-*c*	2ExpTime-*c*
aGRD	*co*NP-*c*	PTime-*c*	2ExpTime-*c*
MFA	2ExpTime-*c*	PTime-*c*	2ExpTime-*c*
RJA	ExpTime-*c*	PTime-*c*	*co*N2ExpTime-*c*
RMFA	2ExpTime-*c*	PTime-*c*	*co*N2ExpTime-*c*

Table 3.2: Summary of complexity of acyclicity conditions

3.3.3 Complexity Analysis of Rules

In this section, we present different complexity results about query answering under existential rules. *Data complexity* is the complexity of query answering when both the query and the set of existential rules are fixed and only the data is given. On the other hand, the *combined complexity* (of query answering) is when the data, the query and the rule set are given. Moreover, the complexity of *membership checking* in a certain class of existential rules or a certain condition $\mathcal{C}$ is for checking whether $\mathcal{C}$ holds given a set of existential rules for an arbitrary database as input.

Table 3.2 summarizes the complexity results related to the above conditions in which c is shorthand for complete.

For example, the hardness result associated with the combined complexity of existential rules that are WA in Table 3.2 is obtained by simulating the behaviour of a 2ExpTime Turing machine by means of a weakly-acyclic set of existential rules. This will immediately lead to 2ExpTime-hardness of BCQ answering under WA sets of existential rules in combined complexity.

Chapter 4

Restricted Chase Termination

In this and the next chapter, we present our first contribution regarding novel classes of decidable rule sets for which the restricted chase terminates. The materials of these two chapters are published in the following venues: Description Logics 2017 [99], International Joint Conference on Rules and Reasoning (RuleML+RR) 2018 [100], and the journal, Theory and Practice of Logic Programming (TPLP) 2020

(invited submission from RuleML+RR 2018).

This chapter is organized as follows. Section 4.1 provides the preliminaries of the chapter, including notations, some basic definitions, and a motivating example. Section 4.2 describes previous work on chase termination, which allows us to compare with the work of the current and the next chapter during its development. Then Section 4.3 sets up the foundation of this work, namely on how to simulate restricted chase for any database by restricted chase with restricted critical databases. We then define in Section 4.4 a hierarchy of classes of finite restricted chase, called k-$\mathsf{safe}(\Phi)$ rule sets for a given cycle function Φ, by testing cycles of increasing nesting depths. We implemented membership checking and a reasoning engine for k-$\mathsf{safe}(\Phi)$ rule sets and conducted experiments. These are reported in Section 4.5.

4.1 Preliminaries

We assume the disjoint countably infinite sets of *constants* C, (*labelled*) *nulls* N, *function symbols* F, *variables* V and *predicates* P. A *schema* is a finite set

$\mathcal{R}$ of relation (or predicate) symbols. Each predicate or function symbol Q is assigned a positive integer as its arity which is denoted by $arity(Q)$. *Terms* are elements in $\mathsf{C} \cup \mathsf{N} \cup \mathsf{V}$. An *atom* is an expression of the form $Q(\mathbf{t})$, where $\mathbf{t} \in (\mathsf{C} \cup \mathsf{V} \cup \mathsf{N})^{arity(Q)}$ and Q is a predicate symbol from $\mathcal{R}$. A *general instance* (or simply an *instance*) I is a set of atoms over the schema $\mathcal{R}$; $term(I)$ denotes the set of terms occurring in I. A *database* is a finite instance I where terms are constants from C. A *substitution* is a function $h : \mathsf{C} \cup \mathsf{V} \cup \mathsf{N} \to \mathsf{C} \cup \mathsf{V} \cup \mathsf{N}$ such that (i) for all $c \in \mathsf{C}$, $h(c) = c$; (ii) for all $n \in \mathsf{N}$, $h(n) \in \mathsf{C} \cup \mathsf{N}$, and (iii) for all $v \in \mathsf{V}$, $h(v) \in \mathsf{C} \cup \mathsf{N} \cup \mathsf{V}$.

Let S_1 and S_2 be sets of atoms over the same schema. A substitution $h : S_1 \to S_2$ is called a *homomorphism* from S_1 to S_2 if $h(S_1) \subseteq S_2$ where h naturally extends to atoms and sets of atoms. In this dissertation, when we define a homomorphism $h : S_1 \to S_2$, if S_1 and S_2 are clear from the context, we may just define such a homomorphism as a mapping from terms to terms.

A rule (also called a *tuple-generating dependency*) is a first-order sentence r of the form: $\forall \mathbf{x} \forall \mathbf{y}\ (\phi(\mathbf{x}, \mathbf{y}) \to \exists \mathbf{z}\ \psi(\mathbf{x}, \mathbf{z}))$, where $\mathbf{x}$ and $\mathbf{y}$ are sets of universally quantified variables (in writing, we often omit the universal quantifier) and ϕ and ψ are conjunctions of atoms constructed from relation symbols from $\mathcal{R}$, variables from $\mathbf{x} \cup \mathbf{y}$ and $\mathbf{x} \cup \mathbf{z}$, and constants from C. The formula ϕ (resp. ψ) is called the *body* of r, denoted $body(r)$ (resp. the *head* of r, denoted $head(r)$). In this chapter and the next chapter, a rule set is a finite set of rules. These rules are also called *non-disjunctive rules* as compared to studies on disjunctive rules (see, e.g., [38, 49]).

We implicitly assume all rules are *standardized apart* so that no variables are shared by more than one rule, even if, for convenience, we reuse variable names in examples provided in this dissertation. A rule is *simple* if variables do not repeat locally inside the body of the rule. A *simple rule set* is a finite set of simple rules.

Given a rule $r = \phi(\mathbf{x}, \mathbf{y}) \to \exists \mathbf{z}\, \psi(\mathbf{x}, \mathbf{z})$, a *Skolem function symbol* f_z is introduced for each variable $z \in \mathbf{z}$, where $arity(f_z) = |\mathbf{x}|$. This leads to the consideration of complex terms, called *Skolem terms*, built from Skolem functions and constants. However, in this dissertation, we will regard Skolem terms

as a special class of nulls (i.e., Skolem terms will be seen as a way of naming nulls).

Ground terms in this context are constants from C or Skolem terms, and atoms in a general instance may contain Skolem terms as well. A *ground instance* in this context is a general instance involving no variables. The *functional transformation* of r, denoted $sk(r)$, is the formula obtained from r by replacing each occurrence of $z \in \mathbf{z}$ with $f_z(\mathbf{x})$. The *Skolemized version* of a rule set R, denoted $sk(R)$, is the set of rules $sk(r)$ for all $r \in R$.

Given a rule $r = \phi(\mathbf{x}, \mathbf{y}) \rightarrow \exists \mathbf{z}\, \psi(\mathbf{x}, \mathbf{z})$, we use $var_u(r)$, $var_{fr}(r)$, $var_{ex}(r)$, and $var(r)$, respectively, to refer to the set of *universal* $(\mathbf{x} \cup \mathbf{y})$, *frontier* $(\mathbf{x})$, *existential* $(\mathbf{z})$, and *all* variables appearing in r. Given a rule set R, the *schema* of R is denoted by $sch(R)$. Given a ground instance I and a rule r, an *extension* h' of a homomorphism h from $body(r)$ to I, denoted $h' \supseteq h$, is a homomorphism from $body(r) \cup head(r)$ to I, that assigns, in addition to the mapping h, ground terms to existential variables of r. A *position* is an expression of the form $P[i]$, where P is an n-ary predicate and i $(1 \leq i \leq n)$ is an integer. We are interested only in positions associated with frontier variables – for each $x \in var_{fr}(r)$, $pos_B(x)$ (resp. $pos_H(x)$) denotes the set of positions of $body(r)$ (resp. $head(r)$) in which x occurs.

We further define that a *path* $(r_1, r_2, \dots)$ (based on R) is a nonempty (finite or infinite) sequence of rules from R; a *cycle* $(r_1, \dots, r_n)$ $(n \geq 2)$ is a finite path whose first and last elements coincide (i.e., $r_1 = r_n$); a *k-cycle* $(k \geq 1)$ is a cycle in which at least one rule has $k + 1$ occurrences and all other rules have $k + 1$ or less occurrences. Given a path π, $\mathsf{Rule}(\pi)$ denotes the set of distinct rules appearing in π.

For a set or a sequence W, the cardinality $|W|$ is defined as usual. The size of an atom $p(\mathbf{x})$ is $|\mathbf{x}|$ and given a rule set R, with $||R||$, we denote the sum of the sizes of atoms in R.

4.2 Chase Variants

The chase procedure is a construction that accepts as input a database I and a rule set R and adds atoms to I which are resulted from applications of rules in R. In this dissertation, our main focus is on the Skolem and the restricted chase variants. However, for the sake of comparison, we introduce the two other main chase variants as well.

We first define triggers, active triggers, and their applications. The Skolem chase is based on triggers, while the restricted chase applies only active triggers.

Definition 23. Let R be a rule set, I an instance, and $r \in R$. A pair (r, h) is called *a trigger for R on I* (or simply *a trigger on I*, as R is always clear from the context) if h is a homomorphism from $body(r)$ to I. If in addition there is no extension $h' \supseteq h$, where $h' : body(r) \cup head(r) \to I$, then (r, h) is called *an active trigger on I*.

An *application* of a trigger (r, h) on I returns $I' = I \cup h(sk(head(r)))$. We write a trigger application by $I\langle r, h\rangle I'$, or alternatively by $I \xrightarrow{\langle r,h\rangle} I'$. We call atoms in $h(body(r))$ the *triggering atoms w.r.t. r and h*, or simply *triggering atoms* when r and h are clear from the context.

Intuitively, a trigger (r, h) is active if given h, the implication in r cannot be satisfied by any extension $h' \supseteq h$ that maps existentially quantified variables to terms in I.

Definition 24. Given a database I and a rule set R, we define the Skolem chase based on a breadth-first fixpoint construction as follows: we let $\mathsf{chase}^0_{sk}(I, R) = I$ and, for all $i > 0$, let $\mathsf{chase}^i_{sk}(I, R)$ be the union of $h(head(sk(r)))$ and $\mathsf{chase}^{i-1}_{sk}(I, R)$ for all rules $r \in R$ and all homomorphisms h such that (r, h) is a trigger on $\mathsf{chase}^{i-1}_{sk}(I, R)$. Then, we let $\mathsf{chase}_{sk}(I, R)$ be the union of $\mathsf{chase}^i_{sk}(I, R)$, for all $i \geq 0$.

Sometimes we need to refer to a *Skolem chase sequence*, which is a sequence of instances that starts from a database I_0 and continues by applying triggers for the rules in a given path on the instance constructed so far. The term *Skolem chase sequence*, in this case, is independent of whether such a sequence

can be extended to an infinite sequence or not. We can also distinguish the two cases where the chase is terminating or not.

A finite sequence of rule applications from a path $(r_1, \ldots, r_n)$ produces a finite sequence of instances $I_0, I_1, \ldots, I_n$ such that

(i) $I_{i-1}\langle r_i, h_i\rangle I_i$, where (r_i, h_i) is a trigger on I_{i-1} for all $1 \leq i \leq n$;

(ii) there is no trigger (r, h) on I_n such that $(r, h) \notin \{(r_i, h_i)\}_{1 \leq i \leq n}$; and

(iii) for each $1 \leq i < j \leq n$, assuming that $I_{i-1}\langle r_i, h_i\rangle I_i$ and $I_{j-1}\langle r_j, h_j\rangle I_j$, $r_i = r_j$ implies $h_i \neq h_j$ (i.e., homomorphism h_i is different from h_j).

The result of the chase sequence is I_n.

An infinite sequence $I_0, I_1, \ldots$ of instances is said to be a *nonterminating Skolem chase sequence* if

(i) for all $i \geq 1$, there exists a trigger (r_i, h_i) on I_{i-1} such that $I_{i-1}\langle r_i, h_i\rangle I_i$;

(ii) for each $i, j \geq 1$ such that $i \neq j$, assuming that $I_{i-1}\langle r_i, h_i\rangle I_i$ and $I_{j-1}\langle r_j, h_j\rangle I_j$, $r_i = r_j$ implies $h_i \neq h_j$.[1]

In this case, the result of the chase sequence is $\bigcup_{i \geq 0} I_i$.

From [113], we know that if some Skolem chase sequence of a rule set R and a database I_0 terminates, then all instances returned by *any* Skolem chase sequence of I_0 and R are terminating and are the same.

Although our technical development does not depend on the oblivious and the core chase which is defined below, we introduce them for a complete background which the reader can safely skip in the first reading.

Before we proceed with the definition of the next chase variant, let us define a notion. The purpose of this notion is to suitably define this version of chase based on Skolemization.

[1] In the literature, in addition to (i) and (ii), another condition known as the *fairness condition for the Skolem chase* is imposed: for each $i \geq 0$, and each trigger (r_i, h_i) on I_{i-1}, there exists some $j \geq i$ such that $I_{j-1}\langle r_i, h_i\rangle I_j$. This last condition guarantees that all the triggers are eventually applied. We remove this requirement, as for the case of the Skolem chase, this condition is immaterial, cf. [75].

Definition 25. Let $r = \phi(\mathbf{x}, \mathbf{y}) \rightarrow \exists \mathbf{z}\, \psi(\mathbf{x}, \mathbf{z})$ be an existential rule. The *oblivious functional transformation* of r, denoted $sk_O(r)$, is the formula obtained from r by replacing each occurrence of $z_i \in \mathbf{z}$ with $f^r_{z_i}(\mathbf{x}, \mathbf{y})$. The *oblivious Skolemized version* of a rule set R, denoted $sk_O(R)$, is defined as: $\bigcup_{r \in R} sk_O(r)$. Let I be an instance and h a homomorphism. An *oblivious application* of a trigger (r, h) on I returns $I' = I \cup h(sk_O(head(r)))$ and is denoted by $I\langle r, h\rangle_O I'$, or alternatively by $I \xrightarrow{\langle r,h\rangle_O} I'$.

The difference between functional transformation of a rule with its oblivious version is that in the former the Skolem function is applied only to frontier variables while in the latter it is applied to universal variables of the rule. Now, we are ready to define the *oblivious chase sequence*.

Definition 26. Let R be a rule set and I_0 a database. A finite sequence $I_0, I_1 \ldots, I_n$, where $n \geq 0$, is called a *terminating oblivious chase sequence* of I_0 w.r.t. R if

(i) for all $0 \leq i < n$, there exists a trigger (r, h) for R on I_i s.t. $I_i\langle r, h\rangle_O I_{i+1}$;

(ii) for each $0 \leq i < j < n$, assuming $I_i\langle r_i, h_i\rangle_O I_{i+1}$ and $I_j\langle r_j, h_j\rangle_O I_{j+1}$, $r_i = r_j$ implies $h_i \neq h_j$; and

(iii) there is no trigger (r, h) for R on I_n s.t. $(r, h) \notin \{(r_i, h_i)\}_{1 \leq i \leq n}$.

The result of the oblivious chase of I_0 w.r.t. R is I_n.

An infinite sequence $I_0, I_1 \ldots$, of instances is called a *nonterminating oblivious chase sequence* of I_0 w.r.t. R if

(i) for all $i \geq 0$, there exists a trigger (r, h) for R on I_i s.t. $I_i\langle r, h\rangle_O I_{i+1}$;

(ii) for each $i, j > 0$ s.t. $i \neq j$, assuming that $I_i\langle r_i, h_i\rangle_O I_{i+1}$ and $I_j\langle r_j, h_j\rangle_O I_{j+1}$, $r_i = r_j$ implies $h_i \neq h_j$.[2]

The result of the oblivious chase of R and I_0 is $\bigcup_{i \geq 0} I_i$.

[2]Again, in the literature, the fairness condition is additionally defined for the oblivious chase similar to that of the Skolem chase.

As opposed to the oblivious chase, the Skolem chase avoids the application of some superfluous triggers. Roughly, if two triggers (r, h_1) and (r, h_2) agree on frontier variables of a given rule r, *i.e.* $h_1|_{var_{fr}(r)} = h_2|_{var_{fr}(r)}$, then they are indistinguishable in the Skolem chase. More formally, let us first define the binary relation $\sim_r$ on the set of homomorphisms $H_r = \{h \mid h : var_u(r) \to (\mathsf{C} \cup \mathsf{T})\}$, where T is the set of all Skolem terms, in a way that $h_1 \sim_r h_2$ iff $h_1|_{var_{fr}(r)} = h_2|_{var_{fr}(r)}$. Then, $\sim_r$ is an equivalence relation on the elements of H_r. So, an oblivious chase sequence $I_0, I_1, \ldots$ is a Skolem chase sequence if for each $i, j \geq 0$, where $i \neq j$, assuming that $I_i \langle r_i, h_i \rangle_O I_{i+1}$ and $I_j \langle r_j, h_j \rangle_O I_{j+1}$, $r_i = r_j = r$ implies $h_i \not\sim_r h_j$.

On the other hand, the restricted chase is known to be order-sensitive. For this reason, it is defined only on sequences of rule applications.

Similar to a Skolem chase sequence, the main idea of a restricted chase sequence (based on a given rule set R) is starting from a given database and applying triggers for the rules in a path based on R on the instance constructed so far. However, unlike the Skolem chase sequence, only active triggers are applied. Similar to the case of the Skolem chase, we distinguish the two cases where the chase is terminating or not.

Definition 27. Let R be a rule set and I_0 a database. A finite sequence $I_0, I_1, \ldots, I_n$ of instances is called *a terminating restricted chase sequence (based on R)* if

(i) for each $1 \leq i \leq n$ there exists an active trigger (r_i, h_i) on I_{i-1} such that $I_{i-1} \langle r_i, h_i \rangle I_i$; and

(ii) there exists no active trigger on I_n.

The result of the chase sequence is I_n.

An infinite sequence $I_0, I_1, \ldots$ is called a *nonterminating (or infinite) restricted chase sequence (based on R)* if

(i) for each $i \geq 0$, there exists an active trigger (r_i, h_i) on I_{i-1} such that $I_{i-1} \langle r_i, h_i \rangle I_i$; and

(ii) it satisfies the *fairness condition*: for all $i \geq 1$ and all active triggers (r_i, h_i) on I_{i-1}, where $r_i \in R$, there exists $j \geq i$ such that either $I_{j-1}\langle r_i, h_i\rangle I_j$ or the trigger (r_i, h_i) is not active on I_{j-1}.

The result of the chase sequence is $\bigcup_{i\geq 0} I_i$.

Example 3. *Let us consider instance* $I = \{P(a,b), P(b,c), P(c,a), Q(a,b)\}$ *and rule* r*:*

$$r : P(x,y), P(y,z), P(z,x) \rightarrow \exists u\, Q(x,u)$$

Homomorphism $h_1 = \{x/a, y/b, z/c\}$ *maps* $body(r)$ *to* I*. The pair* (r, h_1) *is a trigger on* I *and* $I\langle r, h_1\rangle I \cup \{Q(a, f_u(a))\}$ *where* f_u *is a Skolem function constructed from* u*. However,* (r, h_1) *is not active for* I*. On the other hand, homomorphism* $h_2 = \{x/c, y/a, z/b\}$ *maps* $body(r)$ *to* I *and* (r, h_2) *is an active trigger on* I *since there is no extension* h_2' *of* h_2 *such that* $h_2'(head(r)) \subseteq I$*. So, we have* $I\langle r, h_2\rangle I \cup \{Q(c, f_u(c))\}$*. Therefore,* (r, h_1) *can be applied for the Skolem chase but not for the restricted chase, while* (r, h_2) *can be applied for both chase variants.*

Note that the fairness condition essentially says that any active trigger is eventually either applied or becoming inactive. Furthermore, an infinite restricted chase sequence $\mathcal{I}$ cannot be called nonterminating if the fairness condition is not satisfied for $\mathcal{I}$. Recently, in [75], it has been shown that for rules with *single heads* (i.e., where the head of a rule consists of a single atom), the fairness condition can be safely neglected.

A rule set R is said to be *(all-instance) terminating* under the restricted chase, or simply *restricted chase terminating* if it has no infinite restricted chase sequence w.r.t. all databases; otherwise, R is nonterminating under the restricted chase; this is the case where there exists at least one nonterminating restricted chase sequence w.r.t. some database.

The classes of rule sets whose chase terminates on all paths (all possible derivation sequences of chase steps) independent of the given databases (thus all instances) are denoted by $\mathsf{CT}^{\triangle}_{\forall\forall}$, where $\triangle \in \{\mathsf{sk}, \mathsf{res}\}$ (sk for the Skolem chase and res for the restricted chase).

Since a chase sequence is generated by a sequence of rule applications, sometimes it is convenient to talk about a chase sequence in terms of a sequence of rules that are applied. On the other hand, each path can be assigned a chase sequence (which is not unique).

For convenience, given a finite path $\pi = (r_1, \ldots, r_n)$ based on R and database I_0, we say that π *leads to a weakly restricted chase sequence (of R and I_0)* if there are active triggers (r_i, h_i) on I_{i-1} $(1 \leq i \leq n)$ such that $I_{i-1}\langle r_i, h_i\rangle I_i$. Note that the condition is independent of whether there exists an active trigger on I_n or not; so, we do not qualify the sequence $I_0, I_1, \ldots, I_n$ as terminating or nonterminating. Furthermore, the condition only requires the existence of active triggers and does not mention whether the fairness condition is satisfied or not in case the sequence can be expanded to an infinite one. By abuse of terminology, we will drop the word *weakly* in the rest of this dissertation when no confusion arises; this is not a technical concern related to deciding on the finite restricted chase in our approach since our approach is based on certain types of terminating restricted chase sequences.

In what follows, we formally define the core chase.

Definition 28. [61] Let R be a set of rules and I an instance. A parallel chase step is defined from I to some instance J if (i) $I \not\models R$; and (ii) $J = \bigcup_{r \in R, \exists h: var(r) \to term(I) \text{ s.t. } I\langle r,h\rangle K} K$. We write it as: $I \xrightarrow{R} J$. A *core chase step* is defined from I to I' if $I \xrightarrow{R} J$ and $I' = core(J)$. A *core chase sequence* is a sequence of instances $I_0, I_1, \ldots$ such that every instance I_{i+1} in it is obtained from I_i by a core chase step. If a core chase sequence $I_0, I_1, \ldots, I_n$ is finite, then the result of the chase is I_n.

Intuitively, an instance J is obtained from I by simultaneous application of all restricted chase steps and then a core chase step is defined as a parallel chase step, followed by a core computation.

The following inclusion relationships hold for rules with terminating chase written as $CT^X_{Y_1Y_2}$, where $X \in \{obl, sk, res, core\}$ denotes different variants of chase (oblivious, Skolem, restricted and core, respectively) and Y_1 (resp. Y_2) is the quantification over source databases (resp. different chase sequences)

for which the rule set is terminating under X chase.

$$CT^{obl}_{\forall\forall} \subset CT^{sk}_{\forall\forall} \subset CT^{res}_{\forall\forall} \subset CT^{res}_{\forall\exists} \subset CT^{core}_{\forall\forall}$$

Note that all the above inclusions are proper. The following four examples show why this claim holds for each inclusion, respectively, from left to right.

Example 4. *Let $R_1 = \{r_1\}$, where:*

$$r_1 : P(x, y) \rightarrow \exists z P(x, z)$$

For $I = \{P(a, b)\}$, the oblivious chase of I and R_1 does not terminate as witnessed by the following chase sequence: $I_0 = I = \{P(a, b)\} \xrightarrow{\langle r_1, \{x/a, y/b\}\rangle_O}$ $I_1 = I_0 \cup \{P(a, f(a, b))\} \xrightarrow{\langle r_1, \{x/a, y/f(a,b)\}\rangle_O} I_2 = I_1 \cup \{P(a, f(a, f(a, b)))\} \ldots$.

On the other hand, the Skolem chase of I and R_1 terminates trivially.

Example 5. *Let $R_2 = \{r_1, r_2\}$, where:*

$$r_1 : R(x) \rightarrow \exists y\, C(y), P(x, y)$$
$$r_2 : C(x) \rightarrow \exists z\, R(z), P(z, x)$$

For $I = \{R(a)\}$, the Skolem chase of I and R_2 does not terminate and the following chase sequence serves as a witness:
$I_0 = I = \{R(a)\} \xrightarrow{\langle r_1, \{x/a\}\rangle_S} I_1 = I_0 \cup \{C(f^{r_1}_y(a)), P(a, f^{r_1}_y(a))\} \xrightarrow{\langle r_2, \{x/f^{r_1}_y(a)\}\rangle_S}$
$I_2 = I_1 \cup \{R(f^{r_2}_z(f^{r_1}_y(a))), P(f^{r_2}_z(f^{r_1}_y(a)), f^{r_1}_y(a))\} \xrightarrow{\langle r_1, \{x/f^{r_2}_z(f^{r_1}_y(a))\}\rangle_S}$
$I_3 = I_2 \cup \{C(f^{r_1}_y(f^{r_2}_z(f^{r_1}_y(a)))), P((f^{r_2}_z(f^{r_1}_y(a))), f^{r_1}_y(f^{r_2}_z(f^{r_1}_y(a))))\} \ldots$.

However, the restricted chase of I and R_2 terminates for all databases.

Example 6. *Let $R_3 = \{r_1, r_2\}$, where:*

$$r_1 : P(x, y) \rightarrow P(y, x)$$
$$r_2 : P(x, y) \rightarrow \exists z\, P(y, z)$$

For $I = \{P(a, b)\}$, if $(r_1, \langle x/a, y/b\rangle)$ is chosen as the first trigger, the atom $P(b, a)$ is added to I to form $I_1 = I \cup \{P(b, a)\} = \{P(a, b), P(b, a)\}$. From this point on, we can see that there is no active trigger on I_1 and the following path: $P = (r_1, r_2)$ is not a restricted chase path as $P(b, a)$ blocks further active triggers in any restricted chase sequence.

On the other hand, if $(r_2, \langle x/a, y/b\rangle)$ is chosen at the outset, the following infinite (fair) restricted sequence can be constructed:

$I_0 = I = \{P(a,b)\} \xrightarrow{\langle r_2,\{x/a,y/b\}\rangle} I_1 = I_0 \cup \{P(b, f_z^{r_2}(b))\} \xrightarrow{\langle r_1,\{x/a,y/b\}\rangle}$
$I_2 = I_1 \cup \{P(b,a)\} \xrightarrow{\langle r_1,\{x/b,y/f_z^{r_2}(b)\}\rangle} I_3 = I_2 \cup \{P(f_z^{r_2}(b), f_z^{r_2}(f_z^{r_2}(b)))\}$
$\xrightarrow{\langle r_1,\{x/b,y/f_z^{r_2}(b)\}\rangle} I_4 = I_3 \cup \{P(f_z^{r_2}(b), b)\} \xrightarrow{\langle r_1,\{x/b,y/f_z^{r_2}(b)\}\rangle}$
$I_5 = I_4 \cup \{P(f_z^{r_2}(f_z^{r_2}(b)), f_z^{r_2}(f_z^{r_2}(f_z^{r_2}(b))))\} \ldots$. *Therefore, R_3 belongs to $CT^{res}_{\forall\exists}$ and not to $CT^{res}_{\forall\forall}$.*

Example 7. *Let $R_4 = \{r_1\}$, where:*

$$r_1 : P(x) \rightarrow \exists y\, P(y), Q(x)$$

For $I = \{P(a)\}$, the restricted chase of I and R_3 does not terminate as demonstrated by the following chase sequence:
$I_0 = I = \{P(a)\} \xrightarrow{\langle r_1,\{x/a\}\rangle} I_1 = I_0 \cup \{P(f_y^{r_1}(a)), Q(a)\} \xrightarrow{\langle r_1,\{x/f_y^{r_1}(a)\}\rangle}$
$I_2 = I_1 \cup \{P(f_y^{r_1}(f_y^{r_1}(a))), Q(f_y^{r_1}(a))\} \xrightarrow{\langle r_1,\{x/f_y^{r_1}(f_y^{r_1}(a))\}\rangle}$
$I_3 = I_2 \cup \{P(f_y^{r_1}(f_y^{r_1}(f_y^{r_1}(a)))), Q(f_y^{r_1}(f_y^{r_1}(a)))\} \ldots$
Additionally, the core chase of I and R_4 terminates for all databases.

In Table 4.1, complexity of membership checking for different variants of chase is demonstrated [87], in which I denotes a given database instance (i.e., assumes that the input database is given); the first $\forall$ (resp. $\exists$) shows for all (resp. for some) database(s). Furthermore, the second quantification is on chase sequences. From this table, we can observe that membership checking for the restricted (and the core) chase for all database instances is not an RE-complete problem. So, in general, we cannot verify its membership in the corresponding class using a database or a finite set of databases.

<table>
<tr><th>Δ</th><th>$\mathrm{CT}^{\Delta}_{I\exists}$</th><th>$\mathrm{CT}^{\Delta}_{I\forall}$</th><th>$\mathrm{CT}^{\Delta}_{\forall\forall}$</th><th>$\mathrm{CT}^{\Delta}_{\forall\exists}$</th></tr>
<tr><td>Core</td><td colspan="2" rowspan="2"></td><td colspan="2" rowspan="2">co-RE-complete</td></tr>
<tr><td>Restricted</td></tr>
<tr><td>Skolem</td><td colspan="4" rowspan="2">RE-complete</td></tr>
<tr><td>Oblivious</td></tr>
</table>

Table 4.1: Complexity of membership checking for different variants of chase

Finally, a *conjunctive query* (CQ) q is a formula of the form $q(\mathbf{x}) \coloneqq \exists \mathbf{y}\, \Phi(\mathbf{x}, \mathbf{y})$, where $\mathbf{x}$ and $\mathbf{y}$ are tuples of variables and $\Phi(\mathbf{x}, \mathbf{y})$ is a conjunction of atoms with variables in $\mathbf{x} \cup \mathbf{y}$. A *Boolean conjunctive query* (BCQ) is a CQ of the form $q()$. It is well known that, for all BCQs q and for all databases I, $I \cup R \models q$ (under the semantics of first-order logic) if and only if q is entailed by the result of the chase on R and I for either the semi-oblivious or the restricted chase variant [66].

4.2.1 A Concrete Example

To illustrate the practical relevance of the restricted chase and also use it as a running example, let us consider modelling a secure communication protocol where two different signal types can be transmitted: type A for inter-zone communication and type B for intra-zone communication. Let us consider a scenario where a transmitter from one zone requests to establish secure communication with a receiver from another zone in this network. We assume that the number of trusted servers is unknown. Before a successful communication between two users can occur, following a handshake protocol, the transmitter must send a type A signal to a trusted server in the same zone and receive an acknowledgment back. Then, that trusted server sends a type B signal to a trusted server in the receiver zone.

Figure 4.2.1 illustrates the above data transmission scenario where there are just two cells in each of which there are several users (solid dark circles) and base stations (under blue boxes). If a transmitter `t` in cell 1 requests to transmit a data message to a receiver `r` in cell 2, then `t` must establish a handshake protocol to some base station (e.g., `b1`) in the same cell (sending and receiving to/from `b1`). After a handshake protocol is established, `b1` sends a data message to some base station in cell 2 (`b2` in the figure) to complete the required communication before `t` sends a data message to `r`.

Below, we use existential rules to model the required communication protocol (the modeling here does not include the actual process of transmitting signals). Let us assume by default that every server is trusted.

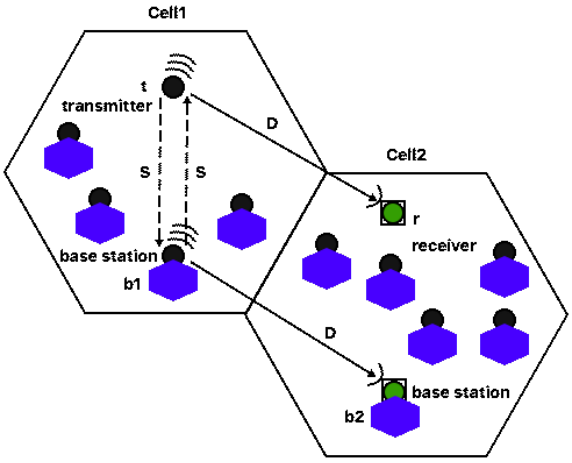

Figure 4.2.1: Data transmission scenario.

Example 8. *Consider the rule set* $R_1 = \{r_1, r_2\}$ *below and its Skolemization, where* $\mathrm{TypeA}(x, y)$ *denotes a request to send a type A signal from* x *to* y *and* $\mathrm{TypeB}(x, y)$ *a request to send a type B signal from* x *to* y*.*

$$r_1 : \mathrm{TypeB}(x, y) \to \exists u\, \mathrm{TypeA}(x, u), \mathrm{TypeA}(u, x)$$
$$r_2 : \mathrm{TypeB}(x, y), \mathrm{TypeA}(x, z), \mathrm{TypeA}(z, x) \to \exists v\, \mathrm{TypeB}(z, v)$$

$$sk(r_1) : \mathrm{TypeB}(x, y) \to \mathrm{TypeA}(x, f_u(x)), \mathrm{TypeA}(f_u(x), x)$$
$$sk(r_2) : \mathrm{TypeB}(x, y), \mathrm{TypeA}(x, z), \mathrm{TypeA}(z, x) \to \mathrm{TypeB}(z, f_v(z))$$

where f_u *and* f_v *are Skolem functions constructed from* u *and* v*, respectively.*

With database $I_0 = \{\mathrm{TypeB}(t, r)\}$*, after applying* $sk(r_1)$ *and* $sk(r_2)$ *under the restricted chase, we get:*

$$I_0 = \{\mathrm{TypeB}(t, r)\} \xrightarrow{\langle sk(r_1), \{x/t, y/r\}\rangle}$$
$$I_1 = I_0 \cup \{\mathrm{TypeA}(t, f_u(t)), \mathrm{TypeA}(f_u(t), t)\} \xrightarrow{\langle sk(r_2), \{x/t, y/r, z/f_u(t)\}\rangle}$$
$$I_2 = I_1 \cup \{\mathrm{TypeB}(f_u(t), f_v(f_u(t)))\}$$

That is, path $\pi_1 = (sk(r_1), sk(r_2))$ *leads to a restricted chase sequence. But this is not the case for the path* $\pi_2 = (sk(r_1), sk(r_2), sk(r_1))$*, since the trigger for applying the last rule on the path is not active – with* $\mathrm{TypeB}(f_u(t), f_v(f_u(t)))$ *as the triggering atom for the body of rule* $sk(r_1)$*, its head can be satisfied by already derived atoms in* I_2*, namely,* $\mathrm{TypeA}(f_u(t), t)$ *and* $\mathrm{TypeA}(t, f_u(t))$ *(i.e., the existential variable* u *in* $sk(r_1)$ *can be instantiated to* t *so that the rule head is satisfied by* I_2*).*

To illustrate more subtleties, let us consider a slightly enriched rule set $R_2 = \{r_3, r_4\}$. *The difference from* R_1 *is that here we use a predicate* $\text{TrustedServer}(a)$ *to explicitly specify that* a *is a trusted server.*

$$r_3 : \text{TypeB}(x, y) \to \exists u\, \text{TrustedServer}(u), \text{TypeA}(x, u), \text{TypeA}(u, x)$$
$$r_4 : \text{TypeB}(x, y), \text{TypeA}(x, z), \text{TypeA}(z, x) \to \exists v\, \text{TrustedServer}(v), \text{TypeB}(z, v)$$

$$sk(r_3) : \text{TypeB}(x, y) \to \text{TrustedServer}(f_u(x)), \text{TypeA}(x, f_u(x)), \text{TypeA}(f_u(x), x)$$
$$sk(r_4) : \text{TypeB}(x, y), \text{TypeA}(x, z), \text{TypeA}(z, x) \to$$
$$\text{TrustedServer}(f_v(z)), \text{TypeB}(z, f_v(z))$$

With the same input database I_0, *we can verify that any nonempty prefix of the 2-cycle* $\sigma = (sk(r_3), sk(r_4), sk(r_3), sk(r_4), sk(r_3))$ *leads to a restricted chase sequence except* σ *itself. Let us provide some details.*

$$I_0 = \{\text{TypeB}(t, r)\} \xrightarrow{\langle sk(r_3), \{x/t, y/r\}\rangle}$$
$$I_1 = I_0 \cup \{\text{TypeA}(t, f_u(t)), \text{TypeA}(f_u(t), t)\} \xrightarrow{\langle sk(r_4), \{x/t, y/r, z/f_u(t)\}\rangle}$$
$$I_2 = I_1 \cup \{\text{TypeB}(f_u(t), f_v(f_u(t)))\}$$

Observe that at this stage, since t *is not known as a trusted server (i.e., we do not have* $\text{TrustedServer}(t)$ *in the given database), unlike the case of* R_1, *we are not able to instantiate the existential variable* u *to* t *to have the rule head satisfied. Thus, the restricted chase continues*

$$I_3 = I_2 \cup \{\text{TrustedServer}(f_u^2(t)), \text{TypeA}(f_u(t), f_u^2(t)), \text{TypeA}(f_u^2(t), f_u(t))\}$$
$$\xrightarrow{\langle sk(r_4), \{x/f_u(t), y/f_v(f_u(t)), z/f_u^2(t)\}\rangle}$$
$$I_4 = I_3 \cup \{\text{TrustedServer}(f_v(f_u^2(t)), \text{TypeB}(f_u^2(t)), f_v(f_u^2(t)))\}$$

Now, the pair $(sk(r_3), \{x/f_u^2(t), y/f_v(f_u^2(t))\})$ *is a trigger on* I_4. *However, since the existential variable* u *in* r_3 *can be instantiated to the Skolem term* $f_u(t)$ *so that the head of* r_3 *is satisfied, the trigger is not active on* I_4 *and thus the chase terminates.*

Figures 4.2.2 and 4.2.3 illustrate the Skolem and the restricted chase on the rule set R_2, where an arrow over a relation symbol indicates a newly derived atom, or an existing atom used to satisfy the rule head so that the restricted chase terminates. In contrast, while R_2 is nonterminating under the Skolem chase, it can be shown that it is all-instance terminating under the restricted chase.

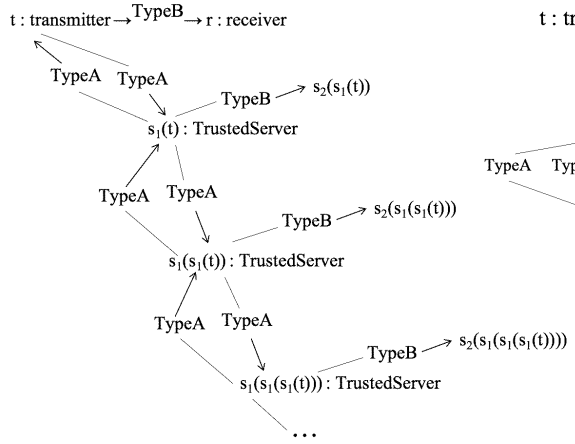

Figure 4.2.2: Skolem chase on R_2.

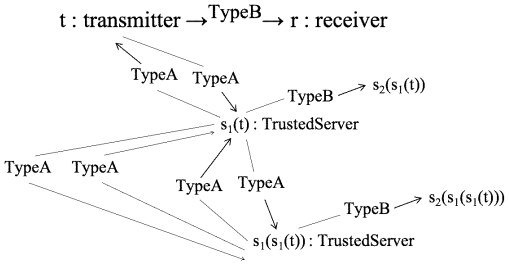

Figure 4.2.3: Restricted chase on R_2.

4.3 Finite Restricted Chase by Activeness

In this section, we tackle the question of what kinds of tests we can do to provide sufficient conditions to identify classes of the finite restricted chase. With this goal in mind, we present the notion of the restricted critical database for a given path and show that any "chained" restricted chase sequence for a given path w.r.t. an arbitrary database can be simulated using the restricted critical database for simple rules and using an *updated restricted critical database* via renaming for arbitrary rules.

4.3.1 Restricted Critical Databases and Chained Property

A primary tool for termination analysis of the Skolem chase is the technique of critical database [113]. Recall that, given a rule set R, if C denotes the set of constants which occur in R, the *critical database* (or *Skolem critical database*) of R, denoted I^R, is a database defined in a way that each relation in I^R is a full relation on the domain $C \cup \{*\}$, in which $*$ is a special constant with no occurrence in R. The critical database can be used to faithfully simulate termination behavior of the Skolem chase – a rule set is all-instance terminating if and only if it is terminating w.r.t. the Skolem critical database. However,

this technique does not apply to the restricted chase.

Example 9. *Given a rule set* $R = \{E(x_1, x_2) \to \exists z\, E(x_2, z)\}$ *and its critical database* $I^R = \{E(*, *)\}$*, where* $*$ *is a fresh constant, the Skolem chase does not terminate w.r.t.* I^R*, which is a faithful simulation of the termination behavior of* R *under the Skolem chase. But the restricted chase of* R *and* I^R *terminates in zero step, as no active triggers exist. However, the restricted chase of* R *and database* $\{E(a, b)\}$ *does not terminate.*

The above example is not at all a surprise, as the complexity of membership checking in the class of rule sets that have a finite restricted chase, namely whether a rule set is in $\mathsf{CT}^{\mathsf{res}}_{\forall\forall}$, is coRE-hard [85], which implies that in general there exists no effectively computable (finite) set of databases which can be used to simulate termination behavior w.r.t. all input databases, as otherwise the membership checking for $\mathsf{CT}^{\mathsf{res}}_{\forall\forall}$ would be recursively enumerable, a contradiction to the coRE-hardness result of [85].

To check for termination, one natural consideration is the notion of cycles based on a given rule set. Firstly, a chase that terminates w.r.t. a database I on all k-cycles implies chase terminating w.r.t. I on all k'-cycles, for all $k' > k$. This is because a chase that goes through a k'-cycle must go through at least one k-cycle. Secondly, since a nonterminating chase must apply at least one rule infinitely many times, if the termination is guaranteed for all k-cycles for a fixed k, then an infinite chase becomes impossible. Thus, testing all k-cycles can serve as a means to decide classes of the finite chase. Furthermore, cycles are recursively enumerable with increasing lengths and levels of nesting. We can test $(k+1)$-cycles for a possible decision of the finite restricted chase when such a test failed for k-cycles. We therefore may find larger classes of terminating rule sets with an increased computational cost. We have demonstrated this approach in Example 8, where the rule set R_2 is terminating on all 2-cycles but not on some 1-cycles. However, a challenging question is *which databases to check against*. In the following, we tackle this question.

Given a path, our goal is to simulate a sequence of restricted chase steps with an arbitrary database by a sequence of restricted chase steps with a fixed

database. On the one hand, since in general we can only expect sufficient conditions for termination, such a simulation should at least capture all infinite derivations by a rule set with an arbitrary database. On the other hand, we only need to consider the type of paths that potentially lead to cyclic applications of the chase. In the following, we will address this question first.

Example 10. *Consider the singleton rule set R with rule $r: T(x,y), P(x,y) \rightarrow \exists z\, T(y,z)$ and its Skolemization $sk(r): T(x,y), P(x,y) \rightarrow T(y, f_z(y))$. With $I_0 = \{T(a,b), P(a,b)\}$, we have: $\mathsf{chase}_{sk}(I_0, R) = I_0 \cup \{T(b, f_z(b))\}$. After one application of r, no more triggers exist and thus the Skolem chase of R and I_0 terminates (so does the restricted chase of R and I_0). This is because the existential variable z in the rule head is instantiated to the Skolem term $f_z(b)$, which is passed to variable y in the body atom $P(x,y)$. As the Skolem term $f_z(b)$ is fresh, no trigger to $P(x,y)$ may be available right after the application of r.*

Note that r in Example 10 depends on itself based on the classic notion of unification. To rule out similar false dependencies, we consider a dependency relation under which the cycle (r, r) in the above example is not identified as a dangerous one. Toward this goal, let us recall the notion of rule dependencies [12][3] and contrast it with its strengthened version for this section.

Definition 29. Let r and r' be two arbitrary rules. Recall that $sk(r)$ and $sk(r')$ denote their Skolemizations.

(i) Given an instance I, we say that r' *depends on* r *w.r.t.* I if there is a homomorphism $h: var_u(r) \rightarrow term(I)$ and a homomorphism $g: var_u(r') \rightarrow term(I) \cup term(h(head(sk(r))))$, such that $g(body(r')) \nsubseteq I$.

(ii) We say that r' *depends on* r if there is an instance I such that r' depends on r w.r.t. I.

If the condition in (ii) is not satisfied, we then say that r' *does not depend on* r, or there is *no dependency from* r' *to* r. Similarly, if the condition in (i)

[3] which was provided earlier for the definition of aGRD in Section 3.3 of Chapter 3.

is not satisfied, we then say that r' *does not depend on* r *w.r.t.* I, or there is *no dependency from* r' *to* r *w.r.t.* I.

The definition in (ii) is adopted directly from [12], which is what a general notion of rule dependency is expected, independent of any instance: r' depends on r if there is a way to apply r to derive some new atoms that are used as part of a trigger to r'. That g is not a homomorphism from $body(r')$ to I requires at least one new atom derived by r, given I. Since instance I can be arbitrary while satisfying the stated condition, no dependency from r' to r means that no matter what the initial database is and what the sequence of derivations is, up to the point of applying r, such an I that satisfies the stated condition does not exist.

By employing an extended notion of unification, the notion of *piece-unification* allows removal of a large number of k-cycles as irrelevant. We will discuss the details in Section 4.5 when we present our experimentation.

The technical focus of rule dependency in this section is the definition in (i), which is strengthened from (ii) by fixing instance I. This is needed because our simulations of the restricted chase are generated from some particular, fixed databases.

Next, we extend the relation of rule dependency to a (nonreflexive) transitive closure. This is necessary since a termination analysis often involves sequences of derivations where rule dependencies yield a transitive relation. Given a path $\pi = (r_1, \ldots, r_n)$, we are interested in a *chain* of dependencies among rules in π such that the derivation with r_n ultimately depends on a derivation with r_1, possibly via some derivations from rules in between. As a chase sequence may involve independent derivations from other rules in between, in the following, we define the notion of projection to reflect this.

Terminology: Given a tuple $V = (v_1, \ldots, v_n)$ $(n \geq 2)$, a *projection of* V *preserving end points*, denoted $V' = (v'_1, \ldots, v'_m)$, is a projection of V (as defined in usual way), with the additional requirement that the end points are preserved (i.e., $v'_1 = v_1$ and $v'_m = v_n$). By abuse of terminology, V' above will simply be called a *projection* of V.

Definition 30. Let R be a rule set, $\pi = (r_1, \ldots, r_n)$ $(n \geq 2)$ a path, and I_0 a database. Suppose that $\mathcal{I} : I_0, I_1, \ldots, I_n$ is a sequence of instances and $H = (h_1, \ldots, h_n)$ is a tuple of homomorphisms such that $I_{i-1}\langle r_i, h_i\rangle I_i$ $(1 \leq i \leq n)$. $\mathcal{I}$ is called *chained* for π if there exists a projection $\mathcal{I}' : I_0, I'_1, \ldots, I'_m$ of $\mathcal{I}$, along with the corresponding projections $H' = (h'_1, \ldots, h'_m)$ of H and $\pi' = (r'_1, \ldots, r'_m)$ of π, such that for all $1 \leq i < m$, r'_{i+1} depends on r'_i w.r.t. I, where $I = I_0$ if $i = 1$ and $I = I'_i \setminus h'_i(head(sk(r'_i)))$, otherwise. If $\mathcal{I}$ is chained for π, we also say that $\mathcal{I}$ has the *chained property*; for easy reference, we sometime also associate the chained property with the corresponding H and say H is chained, or H is a chained tuple of homomorphisms, w.r.t. I_0.

Note that in the definition above, by $I = I'_i \setminus h'_i(head(sk(r'_i)))$, the triggering atoms to r'_{i+1} must include at least one new head atom derived from r'_i.

We now address the issue of which databases to check against for termination analysis of the restricted chase. For this purpose, let us define a mapping $e_i : \mathsf{V} \cup \mathsf{C} \to \langle \mathsf{V}, i\rangle \cup \mathsf{C}$, where constants in C are mapped to themselves and each variable $v \in \mathsf{V}$ is mapped to $\langle v, i\rangle$.

Definition 31. Given a path $\pi = (r_1, r_2, \ldots, r_n)$ of a simple rule set, we define: $I^\pi = \{e_i(body(r_i)) : 1 \leq i < n + 1\}$, which is called a *restricted critical database of* π.

A pair $\langle x, i\rangle$ in I^π is intended to name a *fresh constant* to replace variable x in the body of a rule r_i. The atoms in I^π that are built from these pairs and the constants already appearing in the body of a rule are independent of any given database. The goal is to use these atoms to simulate triggering atoms when necessary, in a derivation sequence from a given database. Let us call these pairs *indexed constants* and atoms with indexed constants *indexed atoms*. Let us use the shorthand v^i for $\langle v, i\rangle$.

Note that due to the structure of I^π, a trigger for each rule in π is automatically available and therefore, without the notion of chained property, a path can rather trivially lead to a restricted chase sequence. To see this, we can construct a restricted chase sequence $I_0, I_1, \ldots, I_n$ based on R as follows.

For each $1 \leq i \leq n$, we construct a trigger (r_i, h_i), where for each variable $v \in var(body(r_i))$, we have $h_i : v \to \langle v, i \rangle$. Since indexed constants are fresh, such a trigger is active.

Example 11. *Consider the rule set R of Example 10 and a path $\pi = (r, r)$. For this rule set, we have $I^\pi = \{T(x^1, y^1),\ P(x^1, y^1), T(x^2, y^2), P(x^2, y^2)\}$. We see that there does not exist any chained tuple of homomorphisms for π w.r.t. I^π. In fact, the claim holds for any instance I since there is no rule dependency from r to r (cf. Definition 29).*

In a restricted critical database that we have seen so far, each body variable is bound to a distinct constant indexed in the order in which rules are applied. Later on, we will motivate and introduce the notion of updated restricted critical databases, where distinct indexed constants may be collapsed into the same indexed constant.

4.3.2 Activeness for Simple Rules

We are ready to define the notion of activeness and show its role in termination analysis for simple rules.

Definition 32. (**Activeness**) Let R be a rule set and I_0 a database. A path $\pi = (r_1 \ldots, r_n)$ based on R is said to be *active* w.r.t. I_0, if there exists a chained restricted chase sequence $\mathcal{I} : I_0, \ldots, I_n$ for π.

The activeness of a path π requires two conditions to hold. First, π must lead to a restricted chase sequence and second, the sequence must have the chained property. In other words, if π is not active w.r.t. I_0, then either some rule in π does not apply due to lack of an active trigger, or the last rule in π does not depend on the first in π transitively in all possible derivations from I_0 using rules in π in that order.

Our goal is to simulate a given chained restricted chase sequence w.r.t. an arbitrary database by a chained restricted chase sequence w.r.t. some fixed databases, while preserving rule dependencies. Such a simulation is called *tight* or *dependency-preserving*. For presentation purposes, we will present the

results in two stages, first for simple rules for which the restricted critical database I^π for a path π is sufficient. Then, in the next subsection, we present the result for arbitrary rules using updated restricted critical databases.

Theorem 33. *Let R be a rule set with simple rules and $\pi = (r_1, \ldots, r_n)$ a path based on R. Then, π is active w.r.t. some database if and only if π is active w.r.t. the restricted critical database I^π.*

Proof. ($\Leftarrow$) Immediate since I^π is such a database.
($\Rightarrow$) Let I be a database w.r.t. which π is active, i.e., there exists a chained tuple of homomorphisms $H = (h_1, \ldots, h_n)$ for π such that (r_i, h_i) $(0 < i \leq n)$ is an active trigger on I_{i-1} and $I_{i-1}\langle r_i, h_i\rangle I_i$. So, there exists a sequence

$$\mathcal{A} : I = I_0, I_1, \ldots, I_n \tag{4.3.1}$$

satisfying the condition: for all $1 \leq i \leq n$, there is a homomorphism $h_i : var_u(r_i) \to term(I_{i-1})$, where $r_i \in R$, such that

$$h_i(body(r_i)) \subseteq I_{i-1}, \tag{4.3.2}$$

$$\forall h_i' \supseteq h_i : h_i'(head(r_i)) \nsubseteq I_{i-1}, \text{ and} \tag{4.3.3}$$

$$I_i = I_{i-1} \cup h_i'(head(r_i)). \tag{4.3.4}$$

We will construct a chained restricted chase sequence of R w.r.t. I^π based on a simulation of derivations in $\mathcal{A}$. Let us denote this sequence by

$$\mathcal{B} : I^\pi = I_0^*, I_1^*, \ldots, I_n^*. \tag{4.3.5}$$

Then, we need to have properties (4.3.2), (4.3.3), and (4.3.4) for $\mathcal{B}$ with h_i and I_{i-1} replaced by some homomorphism g_i and instance I_{i-1}^*, respectively, for all $1 \leq i \leq n$.

To show the existence of such a sequence $\mathcal{B}$, we show how to construct a tuple of homomorphisms $G = (g_1, g_2, \ldots, g_n)$ inductively, such that $I_{i-1}^*\langle r_i, g_i\rangle I_i^*$, for all $1 \leq i \leq n$. This ensures that $\mathcal{B}$ is a Skolem chase sequence. We will then show that all the triggers are active, and along the way, show that G is a chained sequence. We then conclude that $\mathcal{B}$ is, in fact, a chained restricted chase sequence.

Note that instances I_i contain constants from the given database I and instances I_i^* contain indexed constants. Both may contain some constants appearing in rules in π.

We construct g_i along with the construction of a many-to-one function h that maps indexed constants appearing in g_i to constants appearing in h_i. This provides a relation between g_i and h_i. For any atom $a \in body(r_i)$, we call atom $h_i(a)$ an *image* of $g_i(a)$. The function h is many-to-one because distinct indexed constants in g_i may need to be related to a constant in h_i in simulation (in generating sequence $\mathcal{A}$, distinct variables may be bound to the same constant; but in generating sequence $\mathcal{B}$, distinct variables can only be bound to distinct indexed constants).

For $i = 1$, we let $g_1(body(r_1)) \subseteq I^\pi$ with the index in indexed constants being 1. Such g_1 uniquely exists. As (r_1, g_1) is clearly a trigger, we have $I_0^* \langle r_1, g_1 \rangle I_1^*$ under the Skolem chase. For function h, clearly we can let h be such that $h(g_1(a)) = h_1(a)$ for each atom $a \in body(r_1)$.

For any $1 < i \leq n$, we construct g_i as follows. Let $a \in body(r_i)$. If $h_i(a) \in I$, i.e., the triggering atom $h_i(a)$ is from database I, then we let g_i map a to the corresponding indexed atom in I^π with index i. If $h_i(a) \notin I$, i.e., $h_i(a)$ is a derived atom, we then let $g_i(a)$ be any atom whose image is $h_i(a)$.[4] Then, we can extend function h by $h(g_i(a)) = h_i(a)$. Note that this is always possible for simple rules since $body(r_i)$ has no repeated variables. By construction, that (r_i, h_i) is a trigger on I_{i-1} implies that (r_i, g_i) is a trigger on I_{i-1}^*.

We now show that all triggers (r_i, g_i) $(1 \leq i \leq n)$ are active, i.e.,

$$\forall g_i' \supseteq g_i : g_i'(head(r_i)) \nsubseteq I_{i-1}^*, \quad 1 \leq i \leq n \tag{4.3.6}$$

To relate homomorphisms g_i with $\mathcal{B}$ to h_i with $\mathcal{A}$, from above we have $h(g_i(x)) = h_i(x)$, for all $x \in var_u(r_i)$. Then, it follows

$$h(I_{i-1}^*) \subseteq I_{i-1}, \quad 1 \leq i \leq n \tag{4.3.7}$$

which can be shown by induction: for the base case, we have $h(I_0^*) \subseteq I_0$ by

[4]Recall that h is in general many-to-one. So, we may have multiple atoms whose image is $h_i(a)$. Since the rules are assumed to be simple, choosing any of these atoms can lead to the construction of a desired tuple of homomorphisms G as well as the function h.

definition, and for the induction step, for each $k \geq 1$, that $h(I^*_{k-1}) \subseteq I_{k-1}$ implies $h(I^*_k) \subseteq I_k$ is by the construction of homomorphism g_k in $\mathcal{B}$.

To prove (4.3.6), for the sake of contradiction, assume that it does not hold, i.e., $\exists g'_i \supseteq g_i$ s.t. $g'_i(head(r_i)) \subseteq I^*_{i-1}$. This together with (4.3.7) implies $h(g'_i(head(r_i))) \subseteq h(I^*_{i-1}) \subseteq I_{i-1}$. Now let $h'_i(x) = h(g'_i(x))$. It follows $h'_i(head(r_i)) \subseteq I_{i-1}$, a contradiction to (4.3.3). Therefore, all triggers applied in $\mathcal{B}$ are active and π thus leads to a restricted chase sequence of R and I^π.

Finally, $\mathcal{B}$ is chained because the *depends-on* relation in $\mathcal{A}$ is preserved for $\mathcal{B}$. For the path $\pi = (r_1, \ldots, r_n)$, assume that r_j depends on r_i w.r.t. I_{i-1} $(1 \leq i < j \leq n)$. As $\mathcal{A}$ is a restricted chase sequence, we have homomorphisms $h_i : body(r_i) \to I_{i-1}$ and $h_j : body(r_j) \to I_{j-1}$. That r_j depends on r_i w.r.t. I_{i-1} ensures that h_j is not a homomorphism from $body(r_j)$ to $I_{j-1} \setminus h_i(head(sk(r_i)))$. We have already shown the existence of homomorphisms $g_i : body(r_i) \to I^*_{i-1}$ and $g_j : body(r_j) \to I^*_{j-1}$. Since h_j is not a homomorphism from $body(r_j)$ to $I_{j-1} \setminus h_i(head(sk(r_i)))$, it follows by construction that g_j is not a homomorphism from $body(r_j)$ to $I^*_{j-1} \setminus g_i(head(sk(r_i)))$. We, therefore, conclude that r_j depends on r_i w.r.t. I^*_{i-1} $(1 \leq i < j \leq n)$. We are done. □

4.3.3 Activeness for Arbitrary Rules

For non-simple rules, a tight simulation using the restricted critical database I^π for a given path π is not always possible. The following example demonstrates that not all active paths can be simulated.

Example 12. *Consider the following rule set $R = \{r_1, r_2, r_3\}$, where*

$$r_1 : P(x, y) \to Q(x, y)$$
$$r_2 : R(x, y) \to T(x, y)$$
$$r_3 : Q(x, y), T(x, y) \to \exists z\, P(z, x), R(z, x)$$

R is not all-instance terminating since for database $I = \{P(a, b), R(a, b)\}$, there is a nonterminating restricted chase sequence starting from I (assuming that the existential variable z is Skolemized to $f_z(x)$):

$$I_0 = I \qquad I_1 = I_0 \cup \{Q(a, b)\}$$
$$I_2 = I_1 \cup \{T(a, b)\} \qquad I_3 = I_2 \cup \{P(f_z(a), a), R(f_z(a), a)\}$$
$$\ldots\ldots$$

where the corresponding active triggers $(r_1, h_1), (r_2, h_2), (r_3, h_3)$ *can be easily identified. However, as illustrated below, a tight simulation for any path* $\pi = (r_1, r_2, \dots)$ *is not possible for the restricted critical database* I^π*. For example, given* $\pi_1 = (r_1, r_2, r_3)$*, with restricted critical database*

$$I^{\pi_1} = \{P(x^1, y^1), R(x^2, y^2), Q(x^3, y^3), T(x^3, y^3)\}$$

, it is easy to verify that π_1 *is not active w.r.t.* I^{π_1}*. To see why this is the case, consider the following derivation which is obtained after having applied the triggers* (r_1, g_1) *and* (r_2, g_2) *to produce*

$$I_0^* = I^{\pi_1}, \qquad I_1^* = I_0^* \cup \{Q(x^1, y^1)\}, \qquad I_2^* = I_1^* \cup \{T(x^2, y^2)\}$$

The reason that π_1 *is not active w.r.t.* I^{π_1} *is that multiple occurrences of constants* a *and* b *in the triggering atoms on* I_2*, i.e.,* $Q(a, b)$ *and* $T(a, b)$*, are originated from the given database from* different *sources (atoms). For termination analysis, we must provide a simulation of any restricted chase sequence. Below, we discuss two possible solutions using the above example.*

- *Solution 1: Trigger* $(r_3, \{x/x^3, y/y^3\})$ *on* I_2^* *is already available since* $Q(x^3, y^3), T(x^3, y^3) \in I_2^*$*, which can be applied to continue the chase.*
- *Solution 2: Let* rn *be a renaming function that renames indexed constants* x^2 *and* y^2 *appearing in* I^{π_1} *to* x^1 *and* y^1*, respectively, i.e.,* $rn(I^{\pi_1}) = \{P(x^1, y^1), R(x^1, y^1), Q(x^3, y^3), T(x^3, y^3)\}$*, so that* $(r_3, \{x/x^1, y/y^1\})$ *is a trigger on* $rn(I_2^*)$*.*

Solution 1 is rather weak since it allows the simulation of a chained sequence to be "broken" without preserving rule dependency, whereas Solution 2 leads to a tight simulation, i.e., a simulation that preserves the dependency relation of the sequence being simulated. In this chapter and the one that follows, we formalize and develop results for Solution 2.

Given a path π and critical database I^π, let Π_{I^π} be the set of indexed constants appearing in I^π. We define a *renaming function for* I^π to be a mapping from Π_{I^π} to Π_{I^π}. For technical clarity, we eliminate symmetric renaming

functions by imposing a restriction: an indexed constant with index i can only be renamed to an indexed constant with index j, where $1 \leq j < i$. In other words, an indexed constant with index i in I^π can only be renamed to one which appears in a rule in π earlier than r_i.

Theorem 34. *Let R be a rule set and $\pi = (r_1, \ldots, r_n)$ a path based on R. Then, π is active w.r.t. some database if and only if there exists a renaming function rn^* for I^π such that π is active w.r.t. $rn^*(I^\pi)$, where rn^* is composed of at most n renaming functions.*

Proof. ($\Leftarrow$) Immediate since $rn^*(I^\pi)$ is such a database.
($\Rightarrow$) The proof follows the same structure as for Theorem 33 except for the case where the tight simulation of a chase step fails to provide a trigger due to repeated variables in a rule body.

As in the proof of Theorem 33, we assume that path $\pi = (r_1, \ldots, r_n)$ is active w.r.t. some database I, so that there is a chained restricted chase sequence

$$\mathcal{A} : I = I_0, I_1, \ldots, I_n \tag{4.3.8}$$

generated by active triggers $(r_1, h_1), \ldots, (r_n, h_n)$. We show that there exist a renaming function rn^* for I^π and a chained restricted chase sequence w.r.t. $rn^*(I^\pi)$

$$\mathcal{B} : rn^*(I^\pi) = rn^*(I_0^*), rn^*(I_1^*), \ldots, rn^*(I_n^*). \tag{4.3.9}$$

generated by active triggers $(r_1, rn^* \circ g_1), \ldots, (r_n, rn^* \circ g_n))$. We prove the existence of $\mathcal{B}$ by constructing g_i's (and its renamed counterparts) along with the construction of a many-to-one function h that relates indexed constants in g_i (and its renamed counterparts) to constants in h_i. We apply the same argument repeatedly to show the existence of a composed renaming function rn^*. Let us start by constructing the first renaming function, rn_1.

The construction of g_1 is the same as in the proof of Theorem 33 – we let $g_1(body(r_1)) \subseteq I^\pi$ with the index in indexed constants being 1 and let $h(g_1(body(r_1))) = h_1(body(r_1))$. For the inductive case ($1 < i \leq n$), we construct g_i as follows. Let $a \in body(r_i)$. If $h_i(a) \in I$, i.e., the triggering atom

$h_i(a)$ is from database I, then we let g_i map a to a corresponding indexed atom in I^π with index i. If $h_i(a) \notin I$, i.e., $h_i(a)$ is a derived atom, we then consider all body atoms of r_i including a that form a *connected component* in that any two of which share at least one variable. There are in general one or more such connected components in $body(r_i)$. For simplicity and w.l.o.g., let us assume that $body(r_i)$ consists of only one such connected component. If $body(r_i)$ for some $1 \leq i \leq n$ consists of more than one connected component, then we can apply the same techniques used below to construct a sequence of renaming functions – as long as the required properties for the construction of these functions are met for each component (cf. Case (ii) below), the same argument is applicable.

Now let us attempt to construct a mapping g_i by letting $g_i(body(r_i))$ be the set of atoms whose images are precisely those in $h_i(body(r_i))$. There are two cases.

Case (i) g_i is a homomorphism from $body(r_i)$ to I^*_{i-1}. In this case, function h can be extended by $h(g_i(body(r_i))) = h_i(body(r_i))$. By construction, (r_i, g_i) is a trigger on I^*_{i-1}, and the proof that (r_i, g_i) is active remains the same as for Theorem 33.

Case (ii) Otherwise g_i fails to be a homomorphism from $body(r_i)$ to I^*_{i-1}. Assume g_i is the first such failure in the construction of sequence $\mathcal{B}$ so far. Note that the failure is because g_i constructed this way must be a one-to-many mapping – g_i must map a variable to distinct indexed constants because multiple occurrences of a variable in $body(r_i)$ are instantiated to a common constant by h_i but to simulate that, g_i must map the same variable to distinct indexed constants.

The failure can be remedied by a renaming function for I^π, denoted rn_1, by which some different indexed constants are renamed to the same one so that $(r_i, rn_1 \circ g_i)$ is a trigger on $rn_1(I^*_{i-1})$. Clearly, such a renaming function exists. We require that rn_1 be *minimal* in that the number of indexed constants that are renamed to *different ones* is minimized.[5] It is easy to see that the existence

[5]In other words, that an indexed constant is renamed to a different one only when it is

of a renaming function for I^π implies the existence of such a minimal renaming function for I^π. We now want to show that the sequence

$$rn_1(I^\pi) = rn_1(I_0^*), rn_1(I_1^*), \ldots, rn_1(I_{i-1}^*), rn_1(I_i^*) \tag{4.3.10}$$

is a chained restricted chase sequence generated by triggers $(r_1, rn_1 \circ g_1), \ldots, (r_i, rn_1 \circ g_i)$. The function h that relates indexed constants to constants in h_j $(1 \leq j \leq i)$ is updated correspondingly as $h(rn_1 \circ g_j(body(r_j))) = h_j(body(r_j))$.

That $(r_i, rn_1 \circ g_i)$ is a trigger on $rn_1(I_{i-1}^*)$ is by the construction of rn_1. For each $rn_1 \circ g_j$ $(1 \leq j < i)$, since for $rn_1 \circ g_j$ the only update of g_j is that some different indexed constants are replaced by the same one; that g_j is a homomorphism from $body(r_j)$ to I_{j-1}^* implies that $rn_1 \circ g_j$ is a homomorphism from $body(r_j)$ to $rn_1(I_{j-1}^*)$. We now show that triggers $(r_j, rn_1 \circ g_j)$ $(1 \leq j \leq n)$ are all active.

The intuition behind this part of the proof is that in case (i) when we use distinct indexed constants for distinct variables, we do not introduce any possibility of "recycled" atoms (i.e., atoms which can also be used in later derivations). Therefore, the activeness of (r_j, h_j) implies activeness of (r_j, g_j). On the other hand, although the above statement may not hold for case (ii), a renaming function that is minimal ensures that we do not introduce more than what is needed, i.e., $rn_1 \circ g_j$ requires no more mappings to the same constants than h_j. This again ensures that the activeness of trigger (r_j, h_j) implies activeness of trigger $(r_j, rn_1 \circ g_j)$.

More formally, the activeness of $(r_j, rn_1 \circ g_j)$ $(1 \leq j \leq n)$ means that the following conditions hold: for each $1 \leq j \leq n$

$$\forall g_j' \supseteq rn_1 \circ g_j : g_j'(head(r_j)) \nsubseteq rn_1(I_{j-1}^*), \quad 1 \leq j \leq i \tag{4.3.11}$$

We let $h(rn_1 \circ g_j(x)) = h_j(x)$, for all $x \in var_u(r_j)$. Then, by induction we show that

$$h(rn_1(I_{j-1}^*)) \subseteq I_{j-1}, \quad 1 \leq j \leq i \tag{4.3.12}$$

For the base case, we have $h(rn_1(I_0^*)) \subseteq I_0$, which holds due to the minimality of rn_1. For the induction step, for each $k \geq 1$, let us assume $h(rn_1(I_{k-1}^*)) \subseteq$

necessary.

I_{k-1}. Then we need to show $h(rn_1(I_k^*)) \subseteq I_k$. The latter can be done by the construction of $rn_1 \circ g_k$ in (4.3.10).

We then proceed to prove (4.3.11). For this purpose, assume that it does not hold, i.e., $\exists g_j' \supseteq rn_1 \circ g_j$ s.t. $g_j'(head(r_j)) \subseteq rn_1(I_{j-1}^*)$. This together with (4.3.12) implies $h(g_j'(head(r_j))) \subseteq h(rn_1(I_{j-1}^*)) \subseteq I_{j-1}$. Now let $h_j'(x) = h(g_j'(x))$. It follows $h_j'(head(r_j)) \subseteq I_{j-1}$, which is a contradiction to our assumption that (r_j, h_j) for $1 \leq j \leq i$ active. This shows that all triggers applied in (4.3.10) are active.

We then apply the same argument to continue the construction of sequence $\mathcal{B}$ of (4.3.9):

$$rn_1(I^\pi) = rn_1(I_0^*), rn_1(I_1^*), \ldots, rn_1(I_{i-1}^*), rn_1(I_i^*), \ldots \tag{4.3.13}$$

generated by active triggers $(r_j, rn_1 \circ g_j)$ $(1 \leq j \leq i)$ from the updated restricted critical database $rn_1(I^\pi)$. If case (i) applies for the simulation of a chase step in $\mathcal{A}$, then let us use the identity renaming function (which is minimal by definition). Thus, the simulation of each chase step results in a minimal renaming function. It follows that $rn^* = rn_n \circ \cdots \circ rn_1$ and, as the chained property immediately holds by construction, sequence $\mathcal{B}$ is indeed a chained restricted chase sequence. We then conclude that π is active w.r.t. the updated restricted critical database $rn^*(I^\pi)$. We are done. □

In the sequel, given a path π, I^π, and $rn^*(I^\pi)$ for all renaming functions rn^* are all called a restricted critical database. For clarity, we may qualify the latter as an updated restricted critical database.

The development of this section leads to the following conclusion, which can be considered the foundation of our approach to defining classes of the finite restricted chase in this chapter.

Theorem 35. *Let R be a rule set. For any $k > 0$, if no k-cycle σ is active w.r.t. $rn^*(I^\sigma)$, for all renaming functions rn^* for I^σ, then R is all-instance terminating under the restricted chase.*

Proof. Assume that R is not all-instance terminating under restricted chase. Then for some database I_0 there is a nonterminating restricted chase sequence

$\mathcal{I} : I_0, \ldots, I_j, \ldots$. Since I_0 is finite, there can only be a finite number of independent applications of any rule. It follows that $\mathcal{I}$ must contain one chained restricted chase sequence for some k-cycle σ. W.l.o.g., assume that σ appears immediately after an initial, finite segment of $\mathcal{I}$, say $I_0, \ldots, I_i$. It follows that the nonterminating sequence $\mathcal{I}$ without this initial finite segment $\mathcal{I}' : I_i, \ldots, I_j, \ldots$ is a nonterminating chained restricted chase sequence.

By the contraposition of the only if statement of Theorem 34, the assumption that σ is not active w.r.t. $rn^*(I^\sigma)$ for all renaming function rn^* for I^σ, implies that σ is not active w.r.t. any database, i.e., a chained restricted chase sequence for σ does not exist, for any database, which results in a contradiction. □

As we have seen up to this point that renaming enables a tight simulation for termination analysis based on testing k-cycles. A question is whether renaming is a necessary condition in general for our termination analysis. The question is raised due to the following observation.

Example 13. *Consider Example 12 again. We have seen that path* $\pi_1 = (r_1, r_2, r_3)$ *requires renaming in order to obtain a tight simulation. Now consider* $\pi_2 = (r_3, r_2, r_1)$*, which is a permutation of* π_1*. It can be shown that unlike* π_1 *which is not active w.r.t. restricted critical database* I^{π_1}*,* π_2 *is active w.r.t. restricted critical database* I^{π_2}*. According to Theorem 35, as long as there is one k-cycle that is active, we do not conclude that the given rule set is all-instance terminating. For this example, since the 1-cycle* $\sigma = (r_3, r_2, r_1, r_3)$ *is active w.r.t. the restricted critical database* I^σ*, there is no conclusion that* R *is all-instance terminating. This may suggest that if we test all k-cycles, the mechanism of renaming may be redundant. However, the next example shows that this is not the case in general.*

Example 14. *Consider the following rule set* $R' = \{r_1, r_2, r_3\}$ *modified from rule set* R *of Example 12, where*

$$r_1 : P(x, y, z), K(z) \rightarrow Q(x, y, z)$$
$$r_2 : R(x, y, z) \rightarrow T(x, y, z)$$
$$r_3 : Q(x, y, z), T(x, y, z) \rightarrow \exists v\, P(v, x, z), R(v, x, z)$$

R' is not all-instance terminating which can be verified using the database

$$I_0 = \{P(a,b,c), K(c), R(a,b,c)\}$$

We have the following chase sequence starting from database I_0 (assuming that f_v is used to Skolemize the existential variable v) by applying the rules in the path (r_1, r_2, r_3) repeatedly.

$I_1 = I_0 \cup \{Q(a,b,c)\}$, $\quad I_2 = I_1 \cup \{T(a,b,c)\}$,
$I_3 = I_2 \cup \{P(f_v(a,c),a,c), R(f_v(a,c),a,c)\}$, $\quad I_4 = I_3 \cup \{Q(f_v(a,c),a,c)\}$,
.....

The question is: by testing all 1-cycles, can we capture this nonterminating behavior without using renaming? As we show below, the answer is negative.

Similar to the rule set of Example 12, a tight simulation is not possible for any path of the form $\pi = (r_1, r_2, \dots)$ w.r.t. the restricted critical database I^π. However, unlike the rule set of Example 12, no permutation π' of π may lead to a tight simulation for π' w.r.t. the restricted critical database $I^{\pi'}$. For example, consider the path $\pi_2 = (r_3, r_2, r_1)$ which is a permutation of $\pi_1 = (r_1, r_2, r_3)$. The restricted critical database of π_2 is as follows:

$$I^{\pi_2} = \{Q(x^1,y^1,z^1), T(x^1,y^1,z^1), R(x^2,y^2,z^2), P(x^3,y^3,z^3), K(z^3)\}$$

and we derive the following restricted chase sequence:

$$I_0^* = I^{\pi_2}, \quad I_1^* = I_0^* \cup \{P(f_v(x^1,z^1),x^1,z^1), R(f_v(x^1,z^1),x^1,z^1)\}, \quad (4.3.14)$$
$$I_2^* = I_1^* \cup \{T(f_v(x^1,z^1),x^1,z^1)\}, \quad I_3^* = I_2^* \cup \{Q(x^3,y^3,z^3)\}$$

It is easy to check that after derivation of I_2^*, no trigger for r_1 exists that uses atoms derived in I_2^*. Therefore, to derive I_3^*, we have no choice but to pick homomorphism $h = \{x/x^3, y/y^3, z/z^3\}$ to construct trigger (r_1, h) to derive $Q(x^3,y^3,z^3)$. Therefore, the restricted chase terminates since there is no trigger from I_3^*. A similar argument applies to other permutations of π_1. If we conclude that R' is all-instance terminating based on testing all 1-cycles without renaming, we would get a wrong conclusion.

On the other hand, given a (finite) path π, if it leads to a chained restricted chase sequence, starting from the updated restricted critical database $rn^*(I^\pi)$

for some renaming function rn^*, then there is a tight simulation so that π is shown to be active. For example, for $\pi_2 = (r_3, r_2, r_1)$ above we can find an updated restricted critical database of π_2 as follows:

$$rn^*(I^{\pi_2}) = \{Q(x^1, y^1, z^1), T(x^1, y^1, z^1), R(x^2, y^2, z^2), P(x^1, y^1, z^1), K(z^1)\}$$

where indexed constants with index 3 are renamed to those with index 1, so that π_2 is active w.r.t. $rn^*(I^{\pi_2})$.

4.4 K-Safe(Φ) Rule Sets

We now apply the results of the previous section to define classes of the finite restricted chase. The idea is to introduce a parameter, which we call *cycle function*, to generalize various acyclicity notions in the literature. The advantage of this approach is that we can test a path only when it fails to satisfy the given acyclicity condition.

Definition 36. Let R be a rule set and Σ the set of all finite cycles based on R. A *cycle function* is a mapping $\Phi^R : \Sigma \to \{T, F\}$, where T and F denote *true* and $false$, respectively.

Let Φ be the binary function from rule sets and cycles such that $\Phi(R, \sigma) = \Phi^R(\sigma)$, where R is a rule set and σ is a cycle. By overloading, the function Φ is also called a cycle function.

We now address the question of how to obtain a cycle function for an arbitrary rule-based acyclicity condition of finite Skolem chase e.g., JA [105], aGRD [12], MFA [56], etc.

Definition 37. Let Δ denote an arbitrary acyclicity condition of finite Skolem chase (for convenience, let us also use Δ to denote the class of rule sets that satisfy the acyclicity condition expressed by Δ). We define a cycle function Φ_Δ as follows: for each rule set R and each cycle σ based on R, if the acyclicity condition Δ holds for rules in $\mathsf{Rule}(\sigma)$,[6] then Φ_Δ maps (R, σ) to T; otherwise Φ_Δ maps (R, σ) to F.

[6]Recall that $\mathsf{Rule}(C)$ is the set of distinct rules in C.

That is, Φ_Δ maps (R, σ) to T whenever the acyclicity condition Δ for the rule set $\mathsf{Rule}(\sigma)$ is satisfied and to F otherwise. Since any nonterminating restricted chase sequence must involve a cycle of rules, any sufficient condition for acyclicity by definition already guarantees restricted chase termination.

In the sequel, we will use $RS(\Delta)$ to denote the class of rule sets that satisfy the acyclicity condition Δ. Also, according to Definition 37, we will feel free to write $\Phi_\Delta(R, \mathsf{Rule}(\sigma))$ for $\Phi_\Delta(R, \sigma)$.

Example 15. *Consider the rule set R_1 from Example 8 and assume Δ = aGRD in Definition 37. Recall that a rule set R belongs to aGRD (acyclic graph of rule dependencies) if there is no cyclic dependency relation between any two (not necessarily different) rules from R, possibly through other dependent rules of R. Clearly, the corresponding cycle function Φ_Δ maps both cycles $\sigma_1 = (r_1, r_2, r_1)$ and $\sigma_2 = (r_2, r_1, r_2)$ to F.*

We are ready to present our hierarchical approach to defining classes of the finite restricted chase. In the following, we may just write Φ for Φ_Δ as a parameter for cycle functions, or as some fixed cycle function, in particular in a context in which an explicit reference to the underlying acyclicity condition Δ is unimportant.

Definition 38. (k-$\mathsf{safe}(\Phi)$ **rule sets**) Let R be a rule set and σ a k-cycle ($k \geq 1$). We call σ *safe* if for all databases I, σ is not active w.r.t. I. Furthermore, R is said to be in k-$\mathsf{safe}(\Phi)$, or to belong to k-$\mathsf{safe}(\Phi)$ (under cycle function Φ), if for every k-cycle σ which is mapped to F under Φ^R, σ is safe.

For notational convenience, for $k = 0$ we may write 0-$\mathsf{safe}(\Phi_\Delta)$ for $RS(\Delta)$. For example, it can be verified that the rule set R_1 in Example 8 is in k-$\mathsf{safe}(\Phi_\Delta)$ for any $k \geq 1$ and any cycle function Φ_Δ based on some Skolem acyclicity condition Δ in the literature such as weak-acyclicity (WA) [66], Joint-acyclicity (JA) [105], and MFA [56], etc. It is also not difficult to see that the rule set R_2 in the same example belongs to 2-$\mathsf{safe}(\Phi_{\text{aGRD}})$ as well as 2-$\mathsf{safe}(\Phi_{\text{WA}})$ (but note that they do not belong to 1-$\mathsf{safe}(\Phi_{\text{aGRD}})$ or 1-$\mathsf{safe}(\Phi_{\text{WA}})$). However, we stress that R_2 does not belong to any known class of acyclicity,

including RMFA and RJA. That is, rule sets like R_2 are recognized as a finite chase only under the hierarchical framework proposed in the current and the next chapter.

By Theorem 34, k-$\mathsf{safe}(\Phi)$ can be equivalently defined in terms of restricted critical databases.

Proposition 39. *For any cycle function Φ, a rule set R is in k-$\mathsf{safe}(\Phi)$ if and only if every k-cycle σ which is mapped to F under ϕ^R is not active w.r.t. $rn^*(I^\sigma)$, for all renaming functions rn^*.*

We are now in a position to show the following theorem.

Theorem 40. *Let Φ_Δ be a cycle function. For all $k \geq 1$, $(k-1)$-$\mathsf{safe}(\Phi_\Delta) \subseteq k$-$\mathsf{safe}(\Phi_\Delta) \subseteq \mathsf{CT}^{\mathsf{res}}_{\forall\forall}$.*

Proof. For the first subset relation, let us first consider the base case where $k = 1$. Since any nonterminating Skolem chase goes through at least one 1-cycle based on R, if none of the 1-cycles on R violates the corresponding acyclicity condition, i.e., Φ_Δ maps any 1-cycle σ to T, then R trivially belongs to $RS(\Delta)$. Thus, $RS(\Delta) = 0$-$\mathsf{safe}(\Phi_\Delta) \subseteq 1$-$\mathsf{safe}(\Phi_\Delta)$. Then, for all renaming functions rn^*, if there is no chained restricted chase sequence of R and $rn^*(I^\sigma)$ for a k-cycle σ, then there is no chained restricted chase sequence of R and $rn^*(I^{\sigma'})$ for any $(k+1)$-cycle σ', since the latter goes through at least one k-cycle. This shows the first subset relation.

To show the second subset relation, let $R \in k$-$\mathsf{safe}(\Phi_\Delta)$, for any fixed $k \geq 1$. For all k-cycle σ, if (R, σ) is mapped to T by Φ_Δ for every k-cycle σ, then by definition $R \in \mathsf{CT}^{\mathsf{sk}}_{\forall\forall} \subset \mathsf{CT}^{\mathsf{res}}_{\forall\forall}$. If for some k-cycle σ such that (R, σ) is mapped to F by Φ_Δ, then by Proposition 39, $R \in k$-$\mathsf{safe}(\Phi_\Delta)$ implies that σ is not active w.r.t. $rn^*(I^\sigma)$ for all renaming functions rn^* for I^σ. It then follows from inactiveness (Definition 32) and Proposition 39 that there are no chained restricted chase sequences of R and $rn^*(I^\sigma)$. Thus, by Theorem 35, R is restricted chase terminating w.r.t. $rn^*(I^\sigma)$. By the first subset relation, for all $k' > k$, all k'-cycles are terminating. Therefore, we have k-$\mathsf{safe}(\Phi_\Delta) \subseteq \mathsf{CT}^{\mathsf{res}}_{\forall\forall}$. □

Algorithm 4.1 k-safe *Algorithm*

Input: A set of rules R; An integer $k \geq 0$; A cycle function Φ
Output: Boolean value $IsAcyclic$;

```
1: procedure k-safe(R, Φ)
2: end procedure
3: bool IsAcyclic ← true;
4: for each k-cycle σ based on R do
5:     if Φ(R, Rule(σ)) = F then
6:         Find the restricted critical database I^σ;
7:         for each renaming function rn* do
8:             if σ is active w.r.t. rn*(I^σ) then
9:                 return ¬IsAcyclic;
10:            end if
11:        end for
12:    end if
13: end for
14: return IsAcyclic;
```

Finally, we present Algorithm 4.1 to determine whether a rule set belongs to the class k-safe(Φ_Δ). The procedure returns $true$ if it is and $false$ otherwise.

Proposition 41. *Given a rule set R, a cycle function Φ_Δ and an integer $k \geq 1$, R belongs to k-safe(Φ_Δ) if and only if Algorithm k-safe returns true.*[7]

Proof. ($\Rightarrow$) Based on Definition 38, if R is in k-safe(Φ_Δ), then for all k-cycles σ either $\Phi(R, \mathsf{Rule}(\sigma)) = T$, or for all renaming functions rn^* for I^σ, σ is not active w.r.t. restricted critical database $rn^*(I^\sigma)$. Therefore, Algorithm 4.1 returns T.

($\Leftarrow$) By Proposition 39, for all k-cycles σ and for all renaming functions rn^* for I^σ, if σ is not active w.r.t. restricted critical database $rn^*(I^\sigma)$, then the given rule set belongs to k-safe(Φ_Δ), and by Theorem 40, is all-instance terminating.

□

Theorem 42. *Let R be a given rule set and $k \geq 0$ be a unary-encoded integer. Assuming that checking Δ can be done in* PTime*, the complexity of checking membership in k-safe(Φ_Δ) is in* PSpace*.*

[7] The algorithm can be improved by considering only minimal renaming functions, which however would not lower the complexity upper bound. For this reason, we do not pursue the improvement at this level of abstraction.

Proof. Given a rule set R and an acyclicity condition Δ, let us first guess a k-cycle $\sigma = (\sigma_1, \ldots, \sigma_{(k+1)\times|R|-1})$ based on R, and then check whether $\mathsf{Rule}(\sigma) \notin \Delta$. The guessing part can be done using a nondeterministic algorithm. Furthermore, based on our assumption, the checking part can be done in PTime.

For the guessed k-cycle σ, we then proceed by guessing a renaming function rn^* and a restricted chase sequence $\mathcal{I} : rn^*(I^\sigma), \ldots, I_{(k+1)\times|R|-1}$ constructed from σ using a tuple of chained homomorphisms $H = (h_1, \ldots, h_{(k+1)\times|R|-1})$, and verifying whether $\mathcal{I}$ is chained by checking whether σ is active w.r.t. $rn^*(I^\sigma)$, which gives us the complement of the desired membership checking problem.

An iterative procedure is required to construct $\mathcal{I}$. In each step $i > 0$ of this procedure we need to remember each instance I_{i-1} in the constructed sequence, guess a homomorphism h_i, and proceed to derive $I_{(k+1)\times|R|-1}$. For this purpose, we need $\text{NSpace}(((k+1)\times|R|-1)\times\beta)$ memory space to remember intermediate instances, where β is the maximum number of head atoms of rules in σ. In addition, guessing each homomorphism h_i can be done using an NP algorithm and having access to an NP-oracle, verifying if h_i can be extended a homomorphism h_i' and leads to a chained tuple of homomorphisms is NP-complete [130]. All these tasks can be maintained within the same $\text{NSpace}(((k+1)\times|R|-1)\times\beta)$ complexity bound, giving us a $co\text{NSpace}(((k+1)\times|R|-1)\times\beta)$ upper bound for the complexity of membership checking.

As a corollary to Savitch's theorem [131], we have PSpace=NPSpace. Also, based on Immerman–Szelepcsényi theorem [95], nondeterministic space complexity classes are closed under complementation. Therefore, based on the above analysis, the complexity upper bound for the membership checking problem is in PSpace. □

Remark 1. *Based on Theorem 42, it can be seen that for* $\Delta \in \{WA, JA, SWA\}$,[8] *the complexity of checking* k-$\mathsf{safe}(\Phi_\Delta)$ *is in* PSpace. *This shows that our conditions, when considering Skolem acyclicity criteria for which membership checking can be done in* PTime, *are easier to check than even the easiest known condition of the restricted chase in the literature (i.e., RJA) for which*

[8]SWA denotes the *super-weak acyclicity* condition of Skolem chase terminating rule sets [113].

the complexity of membership checking is ExpTime*-complete.*

In addition, for semantic conditions of terminating Skolem chase, such as MSA (respectively, MFA), checking Δ *cannot be done in* PTime *and a worst-case complexity of* ExpTime*-complete (respectively,* 2ExpTime*-complete) can be computed* [88]*. It follows that for the membership checking problem of* k-$\mathsf{safe}(\Phi_\Delta)$*, where* Δ *is MSA (respectively, MFA), an* ExpTime*-complete (respectively,* 2ExpTime*-complete) complexity can be computed. The hardness proof can be established from the membership checking problem of the corresponding class with terminating Skolem chase since this problem cannot be easier than that in general.*

4.5 Experimentation

To evaluate the performance of our proposed methods for termination analysis, we implemented our algorithms in Java on top of the Graal *rule engine* [17]. Our goal was twofold: (1) to understand the relevance of our theoretical approach with real-world applications, and (2) to understand the computational feasibility – even though the problem of checking semantic acyclicity conditions, such as checking activeness of all k-cycles w.r.t. restricted critical databases have a high theoretical worst-case complexity, it may still be a valuable addition to the tools of termination analysis in real-world scenarios.

We looked into a random collection of 700 ontologies from The Manchester OWL Corpus (MOWLCorp) [115], which is a large corpus of ontologies on the web. This corpus is a recent gathering of ontologies through sophisticated web crawls and filtration techniques. After standard transformation into rules (see [56] for details),[9] based on the number of existential variables occurring in transformed ontologies, we picked ontologies from two categories of up to 5 and 5–200 existential variables with equal probability (350 from each). We ran all tests on a Macintosh laptop with 1.7 GHz Intel Core i7 processor, 8GB of RAM, and a 512GB SSD, running macOS Catalina.

[9]Due to limitations of this transformation, our collection does not include ontologies with nominals, number restrictions or denial constraints.

4.5.1 Implementation Setup

Here, we provide the details on our implementation to identify k-$\mathsf{safe}(\Phi_\Delta)$ rule sets.

For a given $k \geq 0$ and a class Δ (which also denotes the corresponding acyclicity condition) of finite Skolem chase, to start, the *candidate pool* of ontologies which is considered for membership in k-$\mathsf{safe}(\Phi_\Delta)$ is the collection of all ontologies. The ontologies that fail our tests for membership in k-$\mathsf{safe}(\Phi_\Delta)$ will be removed. Then at the end of this process, we obtain a set of terminating ontologies.

For each given ontology, we transform it to a rule set R. In our experiments, we consider extending four classes of finite Skolem chase, $\Psi = \{\mathrm{WA}, \mathrm{JA}, \mathrm{aGRD}, \mathrm{MFA}\}$. For each k-cycle σ based on R, first using the technique of piece-unification, we may eliminate R from the candidate pool. If not removed, we then check whether $\mathsf{Rule}(\sigma)$ satisfies the acyclicity condition $\Delta \in \Psi$. If not, we run experiments to check whether σ is active w.r.t. its restricted critical databases.

Let us first introduce the technique based on piece-unification.

Definition 43. (**Piece-unification** [16]) Given a pair of rules (r_1, r_2), a *piece-unifier* of $body(r_2)$ and $head(r_1)$ is a unifying substitution θ of $var(B) \cup var(H)$ where $B \subseteq body(r_2)$ and $H \subseteq head(r_1)$ which satisfies the following conditions:

(a) $\theta(B) = \theta(H)$, and

(b) variables in $var_{ex}(H)$ are unified only with those occurring in B but not in $body(r_2) \setminus B$.

Condition (a) gives a sufficient condition for rule dependency, but it may be an overestimate, which is constrained by condition (b). Note that in Example 10, condition (a) holds for $B = \{T(x, y)\}$ and $H = \{T(y, z)\}$ where $\theta = \{x/y, y/z\}$, and condition (b) does not, since $var_{ex}(H) = \{z\}$ and z unifies with y which occurs in both B and $body(r) \setminus B = \{P(x, y)\}$. Therefore, no piece-unifier of $body(r)$ and $head(r)$ exists.

Piece-unification is known to provide a necessary condition for rule dependencies in that for any two rules r and r', if $body(r)$ and $head(r')$ are not piece-unifiable, then no trigger (r, h) exists that relies on some atom derived from $head(r')$ (cf. Property 18 of [18]). Below, given a substitution θ, $dom(\theta)$ denotes the domain of θ, which is the set of substituted variables in θ, and $codom(\theta)$ denotes the co-domain of θ, which is the set of substitutes in θ. For technical reasons, if θ is a piece-unifier of $body(r)$ and $head(r')$, then $dom(\theta)$ refers to the subset of substituted variables which also appear in $body(r)$ and $codom(\theta)$ refers to the subset of substitutes which appear in $body(r)$ as well.

If the set of all sequences of piece-unifiers that can be constructed from a path π is nonempty, then for each sequence of piece-unifiers that can be formed in π, we need to check whether this sequence leads to a restricted chase sequence or not.

To show whether each sequence of piece-unifiers leads to a sequence of rules which are transitively dependent, checking if they only satisfy conditions (a) and (b) above is not sufficient. Indeed, as shown in [18], given two rules r_1 and r_2, r_2 depends on r_1 if and only if there is a piece-unifier θ of $body(r_2)$ with $head(r_1)$ such that θ satisfies the following conditions: (i) *atom-erasing* and (ii) *productive* (a.k.a. *useful*, cf. [19]). The former condition checks that $\theta(body(r_2))$ is not included in $\theta(body(r_1))$. In addition, the productivity condition for θ means that $\theta(head(r_2))$ is not included in $\theta(body(r_1)) \cup \theta(head(r_1)) \cup \theta(body(r_2))$. Note that the above two conditions can naturally be extended to sequences of piece-unifiers. Therefore, in order to show that each path π does not lead to a chained sequence, it suffices to show that each sequence of piece-unifiers constructed from π (if any), does not satisfy either atom-erasing or productive condition.

The goal of this part is to present how we can eliminate the *irrelevant* k-cycles in our analysis. For this purpose, we utilize the notion of piece-unification as follows, for a given $k > 0$.

- For each k-cycle σ, if the set of sequences of piece-unifiers is $\varnothing$, then σ will be removed from consideration of further checks since σ trivially

leads to a terminating Skolem chase before all the rules in σ are applied (and therefore, a terminating restricted chase).

- For each k-cycle σ, if none of the sequences of piece-unifiers that can be constructed from σ satisfy both conditions of atom-erasing and productive, then σ is removed from our analysis.

We call each k-cycle which has not been removed during the above-mentioned steps, *relevant*.[10]

In our experiments, we performed the following steps:

1. Transforming ontologies in the considered corpus into the normal form using standard normalization techniques (cf. [50]). This will ensure that concepts do not occur nested in other concepts and also each functional symbol introduced during normalization depends on as few variables in the rule as possible. It takes an input ontology path that can be parsed by the OWL API (which is in OWL/XML, OWL Functional Syntax, OBO, RDF/RDFS or Turtle format) (cf. [93]) and produces a normalized ontology; we filter out the following axioms of input ontology: those that are not logical axioms and those containing datatypes, datatype properties, or built-in atoms as the conventional normalization methods are unable to handle them;

2. Rewriting axioms to get first-order logic rules and writing them in the *dlgp format* (for "Datalog+" [14]);

3. Forming all relevant k-cycles Σ constructed from each transformed rule set and for each $\sigma \in \Sigma$, where $\sigma = (r_1, \ldots, r_n)$, we check if $\mathsf{Rule}(\sigma) \in \Delta$, for each Δ from {WA, JA, aGRD, MFA};

4. For each Δ from {WA, JA, aGRD, MFA} and for each relevant k-cycle σ such that $\mathsf{Rule}(\sigma) \notin \Delta$, we check the activeness of σ w.r.t. I^σ, i.e., we

[10]Note that the notion of *compatible unifiers* is introduced in [15] in which piece-unification has been relaxed to take into account arbitrary long sequences of rule applications. This is similar to our goal. In fact, compatible unifiers provide a tighter notion which can help in removing more irrelevant k-cycles.

check if there exists a chained tuple of homomorphisms $H = (h_1, \ldots, h_n)$ for σ at each step $(1 \leq i \leq n)$. We implemented a chained homomorphism checker to accomplish this task;

- During the above check, whenever a relevant k-cycle σ is determined to be active w.r.t. I^σ, the rule set R is removed from the candidate pool;
- If every k-cycle σ is not active w.r.t. I^σ, we check the reason for the failure, say for rule r_i $(1 \leq i \leq n)$. If the failure is due to lack of a trigger which is caused by mapping multiple occurrences of a body variable of r_i to distinct indexed constants, then we know, by Theorem 34, that for some minimal renaming function rn for I^σ, a trigger exists so that there is a chained restricted chase sequence from $rn(I^\sigma)$ up to (and including) r_i. However, we examined all the cases of failure and did not find any failure was caused this way. Therefore, there is no need to continue experiments using the updated restricted critical database as laid out in Theorem 34. This is to say that the phenomenon illustrated in Example 14 did not show up in our collection of practical ontologies.

5. Ontologies in the remaining candidate pool are decided to be terminating.

4.5.2 Experimental Results

For each ontology, we allowed 2.5 hours to complete all of these tasks. In case of running out of time or memory, we report no terminating result. For the first experiment, we considered k-$\mathsf{safe}(\Phi_\Delta)$ rule sets for four different cycle functions Φ_Δ based on WA, JA, aGRD and MFA conditions, respectively, for different values of k.

We consider WA since its acyclicity condition is the easiest to check. We consider three popular syntactic acyclicity conditions WA, JA, and aGRD because the main cost of checking k-$\mathsf{safe}(\Phi_\Delta)$ is then on the extension provided in this chapter. Additionally, we consider MFA, a well known semantic condition

Table 4.2: Membership among 700 ontologies in the collected corpora

k	k-safe(Φ_{aGRD})	k-safe(Φ_{WA})	k-safe(Φ_{JA})	k-safe(Φ_{MFA})
$k = 0$	163	248	299	483
$k = 1$	171	258	310	495
$k = 2$	177	264	316	501
$k = 3$	182	269	321	506
$k = 4$	187	274	326	511
$k = 5$	190	277	329	514
$k = 6$	192	279	331	516

for checking the Skolem chase termination, which is based on forbidding cyclic functional terms in the chase. Note that all other (syntactic) conditions considered in this chapter are subsets of MFA. Besides, it is known that WA $\subset$ JA, and aGRD is not comparable to either WA or JA. We are interested to know whether the high worst-case complexity of our extension prohibits applications in the real world.[11]

In Table 4.2, the results of these experiments are summarized where the values of columns 2–5 denote numbers of ontologies with properties provided in their first row.

Average time analysis for $k = 6$			
Classes	Avg. time (s)	T.W.A.T. (#)	Terminating (%)
6-safe(Φ_{aGRD})	4139	125	27.4
6-safe(Φ_{WA})	3556	164	39.8
6-safe(Φ_{JA})	3231	183	47.2
6-safe(Φ_{MFA})	4923	282	73.7

Table 4.3: Average time analysis for membership testing of terminating ontologies

Consider the case $k = 0$. This is the case where we identify rule sets that are Skolem chase terminating under three acyclicity conditions aGRD, WA, and JA as well as under the MFA condition. First, it is not surprising to observe that among 700 ontologies, the first three syntactic conditions identify only a

[11]For both MFA and RMFA, the complexity of membership checking is already higher than that of Algorithm 1 (assuming checking Δ is in PTIME, cf. Remark 1 in Section 4.4).

small subset of terminating ontologies. However, when considering the MFA condition, we are able to capture many more rule sets as terminating in this collection. Second, for our collection of practical ontologies, the gap between the terminating classes under aGRD and WA conditions is indeed nontrivial. Interestingly, this appears to be the first time that these three syntactic classes of terminating rule sets are compared for practical ontologies. This shows that the theoretical advance from aGRD to WA may have significant practical implications.

As can be seen in Table 4.2, in all of the considered classes, by increasing k, the number of terminating ontologies increases. This is consistent with Theorem 40. Our experiments stopped at $k = 6$ as we did not find more terminating rule sets by testing $k = 7$.

We applied some optimizations in our implementation which will follow. Before proceeding further, let us define some notions. Given a rule set R, consider the graph of rule dependencies $\mathcal{G}_R$ of R in which the set of nodes is R, and there is an edge from some node r_i to a node r_i if r_j depends on r_i. If there is a path from some rule r_i to a rule r_j, then r_i is called to be *reachable* from r_j. If each node in $\mathcal{G}_R$ is reachable from each other node, then $\mathcal{G}_R$ is *connected*. A component of $\mathcal{G}_R$ is a *maximal connected subgraph* of $\mathcal{G}_R$ (i.e., a connected subgraph of $\mathcal{G}_R$ with node set X for which no larger set Y containing X is connected).

For each rule set R, we find the maximal connected subgraphs of $\mathcal{G}_R$ defined as above. Given an acyclicity condition Δ, for each maximal connected subgraph $\mathcal{S}$ of $\mathcal{G}_R$, we check whether $\mathcal{S} \in \Delta$ returns true. If that is the case, then we do not need to check any path based on any nonempty subset of $\mathcal{S}$ for activeness. The reason is that if $\mathcal{S} \in \Delta$, then any subset $\mathcal{S}'$ of $\mathcal{S}$ also satisfies Δ. Therefore, any cycle σ based on $\mathcal{S}'$ is safe. This helped us remove irrelevant subsets of rules in 83 (11.8%) of ontologies in our collection.

Given an acyclicity condition Δ, by Φ_Δ let us denote the cycle function constructed from Δ. Then a notable subclass of 1-$\mathsf{safe}(\Phi_\Delta)$ rules is called $\Delta^<$ introduced in [88] which is defined as the set of rules R in which each simple cycle in the graph of rule dependencies $\mathcal{G}_R$ of R belongs to Δ. In our

collection, it can be seen that only one ontology is $\text{WA}^{\prec}$ (and therefore, $\text{JA}^{\prec}$), which belongs to 1-$\mathsf{safe}(\Phi_{\text{WA}})$ but not in 1-$\mathsf{safe}(\Phi_{\text{aGRD}})$ in Table 4.2.

When k grows from 0, an interesting observation is that for each pair of acyclicity conditions Δ_1 and Δ_2 such that $\Delta_1 \subset \Delta_2$, the rate of increase in the number of terminating rules under Φ_{Δ_2} is faster than that of terminating rules under Φ_{Δ_1} when k grows from $k = 0$ to $k = 1$. Then, for all $k > 1$ the increase of terminating rule sets from $(k-1)$-$\mathsf{safe}(\Phi_\Delta)$ to k-$\mathsf{safe}(\Phi_\Delta)$ is the same for each of the four acyclicity conditions.

Let us see what happens for $k = 1$. Let R be a rule set and Δ_1 and Δ_2 be any pair of acyclicity conditions where $\Delta_1 \subset \Delta_2$. Then there may be some active k-cycles σ based on R such that $\mathsf{Rule}(\sigma) \in \Delta_2 \setminus \Delta_1$. If for all such cycles based on R the above condition holds, then R is in 1-$\mathsf{safe}(\Phi_{\Delta_2})$ but not in 1-$\mathsf{safe}(\Phi_{\Delta_1})$. Hence, for different columns, we see some differences between the numbers added to the first row of Table 4.2 in a way that for any pair of acyclicity conditions Δ_1 and Δ_2 such that $\Delta_1 \subset \Delta_2$, more rules are added as terminating to k-$\mathsf{safe}(\Phi_{\Delta_2})$ compared to k-$\mathsf{safe}(\Phi_{\Delta_1})$, when k increases from 0 to 1.

When $k > 1$, we observe an interesting phenomenon – the same number of terminating rule sets, in fact, the same rule sets, are added. Consider any pair of acyclicity conditions Δ_1 and Δ_2 such that $\Delta_1 \subset \Delta_2$. For any rule set R, assume it is determined to be terminating by Δ_2. In general, more k-cycles based on R need to be checked for the analysis of Algorithm 4.1 in the case of Δ_1 than Δ_2, due to the weaker acyclicity condition in Δ_1. For each such k-cycle σ, since $\text{Rule}(\sigma) \in \Delta_2$, it cannot be active w.r.t. $rn^*(I^\sigma)$ for any renaming function rn^* (otherwise it would contract Theorem 34). Therefore, it must pass the activeness checking of Line 7 in Algorithm 4.1. Consequently, just like how R is determined to be terminating by Δ_2, R is determined to be terminating by Δ_1. Conversely, since the set of k-cycles tested for Δ_1 is a superset of those tested for Δ_2, a rule set R which is determined to be terminating by Δ_1 must also be determined to be terminating by Δ_2. This explains why the number of increases of terminating rule sets for different acyclicity conditions is a constant.

This observation leads to a choice of strategy for testing of k-safe(Φ_Δ) for an expensive acyclicity condition Δ. After failing the check of 1-safe(Φ_Δ), we can check k-safe$(\Phi_{\Delta'})$ for $k > 1$, where Δ' is a weaker but easy-to-check acyclicity condition, with the understanding that the same terminating rule sets will be discovered.

Also, it is clear that if $\Delta_1 \subset \Delta_2$, then for all $k \geq 0$, k-safe$(\Phi_{\Delta_1}) \subset k$-safe$(\Phi_{\Delta_2})$. In Table 4.2, since WA $\subset$ JA $\subset$ MFA, the same inclusion relation holds for k-safe(Φ_Δ) rules constructed from each acyclicity condition Δ for all integers $k \geq 0$.

In order to compare our results with those of [49], we checked the set of terminating ontologies under our conditions for membership in RMFA introduced therein. As a result, it was observed that all the tested rule sets, except for 2 of them, already belong to RMFA. In fact, those two rule sets belong to 6-safe(Φ_{MFA}).

Additionally, we checked the tested corpora for membership in RMFC (cf. [49]). It was observed that 165 (23.6%) of the ontologies belong to RMFC (i.e., for each rule set R in this category, there is a database I_0 for which the restricted chase of R and I_0 is infinite). Based on the above results we find that the termination status of 19 (2.71%) ontologies in the collection is open (i.e., they do not belong to 6-safe(Φ_{MFA}) or RMFC or RMFA). We conducted our tests for membership in RMFA and RMFC using VLog [51].

For the second experiment, we performed time analysis for the tested ontologies for different cycle functions by fixing k to 6. The results are reported in Table 4.3, where the average running time, as well as the number of ontologies *terminating within the average running time* (abbreviated as T.W.A.T.) for that particular cycle function, are reported. It can be seen that in all tested conditions more than half of the terminating ontologies can be determined within the average time. Note that the average times of the table are in seconds.

From our experiments we can see that there is no *one-number-fits-all* k for which any ontology belongs to k-safe(Φ_Δ). However, as observed in our experiments, for real-world ontologies, this number can be indeed small.

TGD generator: For adequately evaluating our approach and also for scalability testing, we implemented a TGD generator on top of [31]. Our goal was to check the performance of implemented classes on large instances with large sets of chase atoms. Our generator can generate custom TGDs while controlling their complexity. It supports an arbitrary number of body atoms. Also, a parameterized total number of predicates and arity of atoms is defined. Furthermore, a parameter is used to control the maximum number of repeated relations in the formula. Each TGD is generated by creating conjunctions and then selecting the subset of atoms that form the head of each TGD.

In our experiments, we generated 500 linear source-to-target TGDs, and 200 linear target TGDs. The reason that we picked linear rules was to control one parameter at a time and also to take the complexities of membership checking under control by focusing on the head atoms to have a better analysis on the restricted chase, as checking activeness of paths is the key here.

For the generated scenarios, we precomputed restricted critical databases for each path based on TGDs resulting from our TGD generator and then, to manage the structure of our TGDs, we tested two different forms of TGD heads: (1) those that have three relations joined in a chain (i.e., the last variable of an atom is joined with the first variable of the next atom which we refer to as *chained TGDs*) and (2) those in which three relations of the head do not share variables (which we refer to as *discrete TGDs*).

In all experiments, each atom has a fixed arity of four, and moreover, each TGD can have up to three repeated relations. The 3 head predicates and the body predicate have been chosen randomly out of a space of 20 predicates. After the generation of each TGD, we check its membership in k-$\mathsf{safe}(\Phi_{\mathrm{WA}})$ for $k = \{0, 1, 2\}$; keep only those TGDs for which this test returns true and discard the rest. Results of different properties in the tested TGDs have been recorded in Tables 4.4 and 4.5.

The results of Tables 4.4 and 4.5 demonstrate that the average running times of membership checking in k-$\mathsf{safe}(\Phi_{\mathrm{WA}})$ for chained TGD generator is more than that of discrete TGD generator. The reason could be in the activeness checking module which takes more time in the rules in which (derived)

Statistics of chained TGD generator for membership in k-$\mathsf{safe}(\Phi_{\mathbf{WA}})$				
k	Avg. time (s)	Terminating (%)	Timeout failure (%)	Memory failure (%)
$k = 0$	54	81	4	0
$k = 1$	135	83.3	6	0.2
$k = 2$	474	86.2	7	0.3

Table 4.4: Statistical results of chained TGD generator for k-$\mathsf{safe}(\Phi_{\mathrm{WA}})$ membership

Statistics of discrete TGD generator for membership in k-$\mathsf{safe}(\Phi_{\mathbf{WA}})$				
k	Avg. time (s)	Terminating (%)	Timeout failure (%)	Memory failure (%)
$k = 0$	43	89	2	0
$k = 1$	94	92	4	0.11
$k = 2$	341	92	4.2	0.16

Table 4.5: Statistical results of discrete TGD generator for k-$\mathsf{safe}(\Phi_{\mathrm{WA}})$ membership

atoms share variables. In addition, in both TGD generators, there are far lesser memory failures than timeout failures.

We performed the same check as detailed in the previous subsection regarding the need for utilizing updated restricted critical databases for rules outputted from our TGD generator, and similar to ontologies in the considered corpora in that section, we did not find any ontology for which we need to run experiments to check activeness with updated restricted critical databases.

Chapter 5

Extension of Bounded Rule Sets

In this chapter, we apply an idea similar to what was introduced in the previous chapter to δ-bounded rule languages of [144], and then, we study membership checking and reasoning complexities. We organize this chapter as follows. In Section 5.1, we define bounded rule sets under the restricted chase variant and compare it with k-safe sets of rules. Then, in Section 5.2, we provide our membership and reasoning complexity analyses for δ-bounded rule sets under the restricted chase. Finally, a discussion regarding relations of our contribution with the most recent papers in this area in the literature along with pointers of how to extend the k-safe hierarchy are presented in Section 5.3.

5.1 Bounded Rule Sets Under the Restricted Chase and Their Connection to K-Safe Hierarchy

In [144], a family of existential rule languages with finite Skolem chase based on the notion of *δ-boundedness* is introduced and the data and combined complexities of reasoning with those languages for *k-exponentially bounded functions* are obtained. Utilizing a parameter called *bound function*, our aim in this section is to show how to extend bounded rule sets from the Skolem to restricted chase. In particular, we show that for any class Δ of terminating rule sets under the Skolem chase, there exists a more general class of terminating rule sets under the restricted chase that extends Δ. We show how to construct such

an extension, and we analyze the membership and reasoning complexities for extended classes. First, let us introduce some terminologies.

A *bound function* is a function from positive integers to positive integers. A rule set R is called *δ-bounded under the Skolem chase* for some bound function δ, if for all databases I, $ht(\mathsf{chase}_{sk}(I,R)) \leq \delta(\|R\|)$, where $\|R\|$ is the number of symbols occurring in R. Given an instance I, $ht(I)$ denotes the height (i.e., maximum nesting depth) of terms that have at least one occurrence in I, if it exists, and ∞ otherwise. In this chapter, when we mention δ as a bound function, we assume that δ is computable.

Let us denote by δ-$\mathcal{B}^{sk}$ the class of δ-bounded rule sets under the Skolem chase. For the restricted case, the definition is similar.

Definition 44. Given a bound function δ, a rule set R is called *δ-bounded under the restricted chase*,[1] denoted δ-$\mathcal{B}^{res}$, if for all databases I and for any restricted chase sequence $\mathcal{I}$ of R and I, $ht(\mathcal{I}) \leq \delta(\|R\|)$.

Example 16. *For the rule set R_1 of Example 8, it can be seen that the height of Skolem terms in all restricted chase sequences is 3. Therefore, R_1 is δ-bounded under the restricted chase variant for some bound function δ for which $\delta(\|R_1\|) = 3$. It is worth noting that R_1 does not belong to δ-bounded rule sets for any computable bound function δ under the Skolem chase.*

Before diving into more details, let us first demonstrate the relationship between δ-bounded rule sets and k-$\mathsf{safe}(\Phi_\Delta)$ rule sets as given in Proposition 45 below.

Proposition 45. *Let R be a k-$\mathsf{safe}(\Phi)$ rule set in which k is a unary encoded integer computable in $\mathcal{O}(P(n))$, for some function $P(n)$. Then R is δ-bounded under the restricted chase for some function δ that is computable in $\mathcal{O}(P(2 \times \log\|R\|))$.*[2]

[1] Note that by definition, the fairness condition is a requirement for a nonterminating restricted chase sequence.

[2] Here n denotes the size of representation for the parameter of k. We say that k can be computed in DTime$(P(n))$ if: There is a deterministic Turing machine M such that, given an integer $l > 0$, M outputs $k(l)$ in $P(\log l)$ stages. Note that $\log l$ is the size of binary representation of l.

Proof. Let R be k-$\mathsf{safe}(\Phi)$. Based on Definition 38, for each k-cycle σ which is mapped to F under Φ, σ is safe (i.e., for all databases I, σ is not active w.r.t. I). Each rule application in a chained sequence can increase the depth of a Skolem term at most by one. Henceforth, the longest possible chained sequence provides an upper bound for the term depth. We show this upper bound is $k \times (k+2)$.

This is because the length of the longest such sequence for a k-cycle is upper bounded by $k \times (k+1)$, and therefore, any sequence of length $k \times (k+2)$ must contain at least one k-cycle. Since no k-cycle is active w.r.t. any database, the depth of any Skolem term generated by the longest chained sequence is less than $k \times (k+2)$. Thus R is $k \times (k+2)$-bounded, which gives a quadratic bound in k. Since k is computable in $\mathcal{O}(P(n))$ and it is unary represented, then k^2 is computable in $\mathcal{O}(P(2 \times \log ||R||))$, where $\log ||R||$ is the size of binary representation of $||R||$. Based on the above argument, we conclude that such a bound function always exists, and $\mathcal{O}(P(2 \times \log ||R||))$ is an upper bound for the cost of computing the bound function. □

5.2 Complexity Analysis for δ-bounded Rule Sets

In what follows, we present our results on the membership of δ-bounded rule sets under the restricted chase variant. Before we proceed, let us define what we mean by membership in the context of this chase version. The problem of membership for the Skolem chase is to check if all Skolem chase sequences halt (terminate) before the maximum height of Skolem terms in each sequence reaches $\delta(||R||)$ for all databases. As described in [144], checking membership for δ-bounded rules under the Skolem chase can be precisely characterized using only one chase sequence and utilizing the Marnette's critical database technique [113], on a single database which is constructed from the given rule set only once.

On the other hand, one cannot determine the membership in the δ-bounded rules under the restricted chase using a single chase sequence. For this purpose,

all possible restricted chase sequences need to be considered. Furthermore, restricted critical databases introduced in Definition 31 can help us determine whether a possible chase sequence constructed from a given rule set witnesses the nonterminating status of the rule set under the restricted chase.

In what follows, we propose a procedure for membership checking of δ-bounded rule sets under the restricted chase. Given a rule set R and a bound function δ, the procedure $MembCheck(R, \delta)$ is defined as follows:

- Check whether R is δ-bounded under the Skolem chase using the (Skolem) critical database constructed from R, denoted I^R. If true, returns T.
- Otherwise, for some $i > 0$, the height of $\mathsf{chase}^i_{sk}(I^R, R)$ is $\delta(||R||) + 1$; for each Skolem chase sequence generated by a path $\pi = (r_1, \ldots, r_n)$ that reaches the height of $\delta(||R||) + 1$, we check whether π is active w.r.t. the restricted critical database $rn^*(I^\pi)$ for all renaming functions rn^*. If the answer is no for all such paths π, then the procedure returns T, otherwise it returns F (false).

A T answer means that R is δ-bounded under the restricted chase and an F answer means that it is unknown whether R is δ-bounded under the restricted chase or not. The reason for the latter case is that when the Skolem chase reaches the height of $\delta(||R||) + 1$ by a path $\pi = (r_1, \ldots, r_n)$, although we can check activeness of π w.r.t. restricted critical databases, we may not be able to determine whether such a path leads to at least one fair sequence.

Proposition 46. *Let δ be a bound function and R an arbitrary rule set. Then $MembCheck(R, \delta)$ is sound, i.e., if it returns T, then R is δ-bounded under the restricted chase. Furthermore, if R consists of rules with single head, then $MembCheck(R, \delta)$ is sound and complete.*

Note that the completeness problem is as follows: $MembCheck(R, \delta)$ is complete if for any given rule set R and bound function δ, if $MembCheck(R, \delta) = F$, then R is not δ-bounded under the restricted chase.

Proof. Let δ be a bound function. By [113], it suffices to use the Skolem critical database I^R to capture all Skolem chase sequences w.r.t. any database I,

so that $ht(\mathsf{chase}_{sk}(I,R)) \leq \delta(\|R\|)$ only if $ht(\mathsf{chase}_{sk}(I^R,R)) \leq \delta(\|R\|)$. Consequently, if R is δ-bounded under the Skolem chase w.r.t. I^R, it is δ-bounded under the Skolem chase w.r.t any database I, and by the relationship between the Skolem and restricted chase, R is δ-bounded under the restricted chase w.r.t any database I.

Otherwise, for each path π that leads to some Skolem chase sequence that reaches the height of $\delta(\|R\|)+1$, π being not active w.r.t. $rn^*(I^\pi)$ for all renaming function rn^* for I^π implies, by Theorem 34, that π is not active w.r.t. any database. When all chained sequences of path π fail to reach the height of $\delta(\|R\|)+1$, no restricted chase sequence of π can reach that height because an unchained sequence does not expand Skolem terms cumulatively throughout. It follows that the largest height by any database is bounded by $\delta(\|R\|)$. This gives the desired conclusion for the soundness of $MembCheck$ for arbitrary rules.[3]

For any single head rule set R, from [75], we know that the fairness condition can be safely neglected, i.e., the existence of a (possibly unfair) infinite restricted chase sequence implies the existence of a fair one. Therefore, R is not δ-bounded. □

Proposition 47. *Let R be a rule set and δ a bound function computable in* $\mathrm{DTime}(P(n))$[4] *for some function $P(n)$. Then, it is in*

$$co\mathrm{NTime}(\mathsf{C}_\delta + \|R\|^{\|R\|^{\mathcal{O}(\delta(\|R\|))}})$$

to check if $MembCheck(R,\delta)$ returns T, where $\mathsf{C}_\delta = P(\log\|R\|)^{\mathcal{O}(1)}$.

Proof. For the Skolem chase with Skolem critical database, from Proposition 6 of [144] we know that using the critical database technique of [113], the max-

[3]If there exists such a path π that is active and leads to a restricted chase sequence, which by default must be fair, then we can decide that R is not δ-bounded under the restricted chase (again, the fairness condition must be satisfied). In this case, the procedure is complete by returning F. On the other hand, if all such paths π lead *only* to unfair restricted chase sequences (i.e., infinite chase sequences generated by active triggers in Definition 27 without requiring the fairness condition), then no restricted chase sequence has reached beyond the bound and in this case, that our procedure returns F shows its incompleteness. But in general, the problem of whether such a π leads *only* to unfair chase sequences may be undecidable.

[4]The class of complexity languages decidable in time $P(n)$ using a deterministic Turing machine. NTime is defined similarly but using a nondeterministic Turing machine.

imum number of atoms generated in a Skolem chase sequence is bounded by $||R||^{||R||^{\mathcal{O}(\delta(||R||))}}$, which is also an upper bound for the number of atoms generated in a restricted chase sequence.

From [113] we know that in the case of the Skolem chase if any sequence terminates on a rule set R and a database I, then the instances returned by all sequences are isomorphically equivalent. So, for δ-boundedness for the Skolem chase, it suffices to consider only one sequence. But for the case of the restricted chase, we need to consider all such sequences.

Given a rule set R and a bound function δ, the procedure $MembCheck(R, \delta)$ first checks whether R is δ-bounded under the Skolem chase.

For the complexity of this check, we need to consider the size of each Skolem chase sequence to produce the height of $\mathcal{O}(\delta)$ that is upper bounded by $||R||^{||R||^{\mathcal{O}(\delta(||R||))}}$, which can be computed in $\text{DTIME}(||R||^{||R||^{\mathcal{O}(\delta(||R||))}})$. In addition, an upper bound for the chase of size $||R||^{||R||^{\mathcal{O}(\delta(||R||))}}$ can be computed in $\text{DTIME}\big((||R|| + P(\log ||R||))^{\mathcal{O}(1)}\big)$. Therefore, according to [144], the overall complexity of this check is: $\text{DTIME}\big((P(\log ||R||))^{\mathcal{O}(1)} + ||R||^{||R||^{\mathcal{O}(\delta(||R||))}}\big)$.

If the above condition is not satisfied (i.e., some R is not δ-bounded under the Skolem chase), for some $i \geq 1$, the height of $\mathsf{chase}_{sk}(I^R, R)$ is $\delta(||R||) + 1$. So, for each Skolem chase sequence that is generated by a path $\pi = (r_1, \ldots, r_n)$ which reaches the height of $\delta(||R||) + 1$, for all renaming functions rn^* for I^π, we check whether π is active w.r.t. $rn^*(I^\pi)$. A *no* answer to the above check yields a T output from $MembCheck(R, \delta)$.

Based on the above argument, to proceed, using a nondeterministic algorithm we first guess a sequence of triggers $\bigcup_{i=1}^{N}(r_i, h_i)$, where N is upper bounded by $||R||^{||R||^{\mathcal{O}(\delta(||R||)+1)}} = ||R||^{||R||^{\mathcal{O}(\delta(||R||))}}$ that can lead to the construction of a Skolem chase sequence $\mathcal{I}$, and a renaming function rn^*.

Then we need to verify if $\mathcal{I}$ is active w.r.t. $rn^*(I^\pi)$, where π is the path constructed from the guessed r_i's. For the latter, for each projection π' of π, first, to verify the chained property, we determine if each rule in π' depends on some previous rule in the path. The complexity of this latter verification task is quadratic in the size of the guessed chase sequence.

Furthermore, given path π, the maximum number of chained restricted

chase sequences is bounded by $||R||^{\mathcal{O}(\delta(||R||))}$, and since the length of the guessed sequence is bounded by $\mathcal{O}(||R||^{||R||^{\mathcal{O}(\delta(||R||))}})$, verifying if $\mathcal{I}$ is active w.r.t. $rn^*(I^\pi)$ is at most polynomial in $||R||^{||R||^{\mathcal{O}(\delta(||R||))}}$ which can be implemented in $\text{NTime}(||R||^{||R||^{\mathcal{O}(\delta(||R||))}})$. Similar to the proof of Theorem 42, the construction of renaming functions can take at most polynomial in the size of π which can be done in $\text{NTime}(||R||^{||R||^{\mathcal{O}(\delta(||R||))}})$. So, clearly, all the above tasks can be maintained in $\text{NTime}\big((P(\log||R||))^{\mathcal{O}(1)} + ||R||^{||R||^{\mathcal{O}(\delta(||R||))}}\big)$.

The membership is complement to the above problem, and therefore, belongs to $co\text{NTime}(\mathsf{C}_\delta + ||R||^{||R||^{\mathcal{O}(\delta(||R||)}})$ as desired. □

Next, we investigate membership and reasoning complexities of bounded rule sets under what is called *exponential tower functions*, which are defined as follows:

$$\mathsf{exp}_\kappa(n) = \begin{cases} n & \kappa = 0 \\ 2^{\mathsf{exp}_{\kappa-1}(n)} & \kappa > 0 \end{cases}$$

Since the complexity of checking δ-bounded property of Proposition 47 is dominated by the second term inside $co\text{NTime}$, if $\delta(n) = \mathsf{exp}_\kappa(n)$, then its overall complexity increases by two exponentials. We thus have

Corollary 48. *Given a rule set R checking if $MembCheck(R, \mathsf{exp}_\kappa)$ returns T is in $co\text{N}(\kappa+2)$-ExpTime.*

Example 17. *Based on the observation made in Example 16, the rule set R_1 in Example 8 is exp_0-bounded under the restricted chase; however, it does not belong to exp_κ-bounded ontologies under the Skolem chase for any computable κ.*

Data and Combined Complexity:

Now, let us investigate the reasoning complexities. The problem under consideration is Boolean Conjunctive Query (BCQ) answering which is defined as follows. Given a rule set R, a database I and a Boolean query q, decide if $I \cup R \models q$. The complexity of this problem is also known as *combined complexity* since the input size is the combined size of all I, R, and q. In the

BCQ answering problem if R and q are fixed and only I changes, then it is called *data complexity*. Focusing on exp_κ-bounded rule sets under the restricted chase variant, we have the following results on reasoning complexities.

Theorem 49. *The problem of Boolean conjunctive answering for* exp_κ*-bounded rule sets under the restricted chase variant is* $(\kappa+2)$-ExpTime*-complete for combined complexity and* PTime*-complete for data complexity.*

Proof. Let R be an exp_κ-bounded rule set under the restricted chase variant and I be a database. Then, let us guess a restricted chase sequence $\mathcal{I}$ nondeterministically. With an argument similar to that of the proof of Proposition 47 in which $\delta(n) = \mathsf{exp}_\kappa(n)$, we know that the number of atoms of $\mathcal{I}$ is bounded by $||R||^{||R||^{\mathsf{exp}_\kappa(||R||)}} = \mathcal{O}(\mathsf{exp}_{\kappa+2}(||R||))$.

The membership follows since the entailment of a BCQ q can be shown by finding such a sequence $\mathcal{I} : I = I_0, \ldots, I_n$ based on R such that I_n satisfies q according to the following fact from [65]: Let J and K be two finite instances returned by the restricted chase of an exp_κ-bounded rule set R and a database I. Then K and J are homomorphically equivalent. Based on the above fact and the homomorphic equivalence classes, in the rest of this proof, we let $chase_{res}(I, R)$ denote one representative of the equivalence class for all results of the restricted chase of R and I. In addition, based on [65], it is known that $chase_{res}(I, R) \models q$, and also there is a homomorphism from I to $chase_{res}(I, R)$. Furthermore, $I \cup R \models q$ if and only if $chase_{res}(I, R) \models q$.

Let k and n denote the number of relation symbols and the maximal arity of relation symbols appearing in R, respectively. Let further l and m represent the number of function symbols, and the maximal arity of function symbols appearing in $sk(R)$, respectively. In addition, let c denote the number of constants appearing in I, and $Q(\mathbf{t})$ be a fact in $chase_{res}(I, R)$. It is easy to verify that the number of symbols in each constituent $t \in \mathbf{t}$ is upper bounded by $\sum_{i=0}^{\mathsf{exp}_\kappa(||R||)} m^i = m^{\mathcal{O}(\mathsf{exp}_\kappa(||R||))}$. Also, it is clear that each symbol is either a constant or a function symbol. Therefore, the number of facts in $chase_{res}(I, R)$ is upper bounded by $(c+l)^{m^{\mathcal{O}(\mathsf{exp}_\kappa(||R||))} \times n} \times k$. Since $k, n, l, m \leq ||R||$, and $c = |dom(I)|$, the following upper bound is derived for the number of facts in

$chase_{res}(I,R)$: $(|dom(I)|+||R||)^{||R||^{\mathcal{O}(\mathsf{exp}_\kappa(||R||))}\times||R||^{\mathcal{O}(1)}}$, which can be computed in $\textsc{DTime}\big((|dom(I)|+||R||)^{||R||^{\mathcal{O}(\mathsf{exp}_\kappa(||R||))}}\big)$.

To compute the reasoning complexity involving a BCQ q, it is now sufficient to evaluate q on $chase_{res}(I,R)$ directly.[5] To continue the analysis, we only need the number of existential variables occurring in q, which we denote by v. Then we need to check whether there is a substitution h which maps every existential variable in q to a ground term of height less than $\mathsf{exp}_\kappa(||R||)$, such that $h(q) \subseteq chase_{res}(I,R)$. From the previous analysis, it is clear that $(|dom(I)|+||R||)^{||R||^{\mathcal{O}(\mathsf{exp}_\kappa(||R||))}\times v}$ substitutions need to be checked. Since $v \leq ||q||$, the evaluation of checking whether $h(q) \subseteq chase_{res}(I,R)$ can be done in

$$\textsc{DTime}\big((|dom(I)|+||R||)^{||R||^{(\mathsf{exp}_\kappa(||R||)\times||q||^{\mathcal{O}(1)}}}\big)$$

Hence, a $(\kappa+2)$-ExpTime upper bound can be computed for the combined complexity, as desired.

We can use a construction similar to that of [144] for the hardness proof. We briefly sketch it here. Let us consider a deterministic Turing machine M which terminates in $\mathsf{exp}_{\kappa+2}(n)$ number of steps on any input of length n. Let us assume that the query and data schema is a singleton set $\{\mathsf{Accept}\}$ and $\varnothing$, respectively, where Accept is a nullary relation symbol. We need to show that for each input x that is a binary string of length n, there is an exp_κ-bounded rule set under the restricted chase variant such that M terminates on x if and only if $\varnothing \cup R \models \mathsf{Accept}$. To construct the rule set R, we need to define a linear order of length $\mathsf{exp}_{\kappa+2}(n)$ on integers which are represented in binary strings from 0 to $\mathsf{exp}_{\kappa+2}(n)$. Once a linear order is defined, we can construct a set of existential rules to encode the Turing machine M and the input x. Once we have such a construction, we can establish the lower bound on the combined complexity of reasoning with existential rules under the restricted chase. This lower bound combined with the upper bound derived above provides the exact bound for the combined complexity of exp_κ-bounded rule sets under the restricted chase.

Furthermore, the data complexity of query answering with exp_κ-bounded

[5]Without loss of generality, we assume that q is in *prenex normal form*.

rule sets under restricted chase is PTIME-complete. The PTIME upper bound for the data complexity can be derived from the above analysis, and the hardness follows from the PTIME-completeness of data complexity of Datalog, cf. [59]. □

5.3 Discussion

In this section, we introduce some more recent papers in this area and then show how to leverage them to extend our proposed k-$\mathsf{safe}(\Phi)$ classes uniformly. In [104], it is shown that there are examples of TGDs for which the data complexity of the restricted chase can reach nonelementary upper bounds. Note, however, that as shown in [144], given any $\kappa > 0$, for any exp_κ-bounded rule set R under the Skolem chase variant, the Boolean query answering problem is PTIME-complete for the data complexity. Therefore, the restricted chase can *realize* queries which are out of the reach for the Skolem chase variant.

Let us define the notion of a *strategy* as a plan of choosing paths based on a given rule set. Utilizing this notion allows us to focus on a *concrete plan for path selection* in the course of our termination analysis for the restricted chase to extend the set of terminating rules under the restricted chase. Exploiting the above terminology, $\mathsf{CT}^{\mathsf{res}}_{\forall\forall}$ can be alternatively defined to be the set of rules with *terminating restricted chase for all strategies* and all instances.

On the other hand, from [121] it is known that $\mathsf{CT}^{\mathsf{res}}_{\forall\forall} \subset \mathsf{CT}^{\mathsf{res}}_{\forall\exists}$, where $\mathsf{CT}^{\mathsf{res}}_{\forall\exists}$ denotes the class of rule sets R such that for all instances I there exists at least one restricted chase sequence of I and R that is finite. Similarly, we can define $\mathsf{CT}^{\mathsf{res}}_{\forall\exists}$ to be the set of rules with *terminating restricted chase for some strategy* and all instances.

Recently, a chase variant known as the *Datalog-first chase* has been introduced in [49] and subsequently in [104], which extends all-path restricted chase by focusing on a particular class of strategies that prioritizes the application of non-generating (Datalog) rules in any considered restricted chase sequence. Let $\mathsf{CT}^{\mathsf{dlf}}_{\forall\forall}$ denote the set of rules with a terminating Datalog-first chase for

all strategies (paths) and all instances.[6] Then we have $\mathsf{CT}^{\mathsf{res}}_{\forall\forall} \subset \mathsf{CT}^{\mathsf{dlf}}_{\forall\forall} \subseteq \mathsf{CT}^{\mathsf{res}}_{\forall\exists}$. Note that although the first inclusion is strict, at the time of writing this dissertation, it is not known whether the second inclusion above is also strict or not.

Based on what was discussed above, we can extend the set of δ-bounded rules under the restricted chase variant as well as k-$\mathsf{safe}(\Phi)$ rules for a given bound function δ, integer k and cycle function Φ by considering the Datalog-first chase. For this purpose, we only need to focus on cycles in which applications of Datalog rules are prioritized. Given a bound function δ, let us call any rule set R that is δ-bounded under the above condition *δ-bounded under the Datalog-first chase*. We can define *k-$\mathsf{safe}(\Phi)$ rules under the Datalog-first chase* similarly.[7]

Example 18. *Let $R = \{r_1, r_2\}$ (adopted from [75]), where*

$$r_1\colon\ Q(x,y,y) \to \exists u Q(x,u,y), Q(u,y,y)$$
$$r_2\colon\ Q(x,y,z) \to Q(z,z,z)$$

Note that in this rule set the fairness condition requires application of r_2 in any (fair) sequence of the restricted chase and after r_2 is applied, the next application of r_1 is not active and therefore, any (fair) restricted chase sequence terminates. The following derivation starting from $\{Q(a,b,b)\}$ demonstrates such a sequence in which fresh nulls z_i are used to instantiate the existential variable u:

$$I_0 = \big\{Q(a,b,b)\big\} \xrightarrow{\langle r_1,\{x/a,y/b\}\rangle}$$
$$I_1 = I_0 \cup \big\{Q(a,z_1,b), Q(z_1,b,b)\big\} \xrightarrow{\langle r_1,\{x/z_1,y/b\}\rangle}$$
$$I_2 = I_1 \cup \big\{Q(z_1,z_2,b), Q(z_2,b,b)\big\} \xrightarrow{\langle r_1,\{x/z_2,y/b\}\rangle}$$
$$I_3 = I_2 \cup \big\{Q(z_2,z_3,b), Q(z_3,b,b)\big\} \xrightarrow{\langle r_1,\{x/z_3,y/b\}\rangle}$$
$$\dots$$
$$I_{j-1} = I_{j-2} \cup \big\{Q(z_{j-2},z_{j-1},b), Q(z_{j-1},b,b)\big\} \xrightarrow{\langle r_2,\{x/a,y/b,z/b\}\rangle}$$
$$I_j = I_{j-1} \cup \big\{Q(b,b,b)\big\}$$

[6] Note that by strategy in the Datalog-first chase we mean a plan for choosing the application order of Datalog rules which must always occur before the application of generating rules (i.e., non-full TGDs).

[7] In this case, R is said to be in k-$\mathsf{safe}(\Phi)$ under the Datalog-first chase, or to belong to k-$\mathsf{safe}(\Phi)$ under the Datalog-first chase (given a cycle function Φ and an integer k), if for every k-cycle σ which prioritizes Datalog rules (except the last rule of the cycle), and is mapped to F under Φ^R, σ is safe.

Note that in the above sequence of derivations, the following step: $I_j\langle r_1, \{x/z_{j-1}, y/b\}\rangle I_{j+1}$ *does not exist, and any valid restricted chase sequence terminates. However, the fairness condition needs the existence of some* j *to apply some active trigger involving* r_2 *(i.e.,* $\langle r_2, \{x/a, y/b\}\rangle$ *in this example). But due to the nondeterministic nature of this process,* j *can be chosen anywhere in the sequence.*

As discussed above, the rule set R *in this example is not* δ*-bounded under the restricted chase for any computable bound function* δ*. However, starting from any database* I*, no (fair) infinite restricted chase sequence can be constructed from* R *and* I*.*

On the other hand, it is not hard to see that R *belongs to* $1\text{-}\mathsf{safe}(\Phi_{WA})$ *under the Datalog-first chase. The reason is that this chase variant requires the application of* r_2 *before* r_1 *in any valid chase sequence. Therefore, all* 1*-cycles in which the application of Datalog rules are prioritized (i.e.,* (r_2, r_1, r_2)*) are safe. The following sequence of derivations shows why this is the case.*

$$
\begin{aligned}
&I'_0 = \big\{Q(a,b,c)\big\} \xrightarrow{\langle r_2, \{x/a, y/b, z/c\}\rangle} \\
&I'_1 = I'_0 \cup \big\{Q(b,b,b)\big\} \xrightarrow{\langle r_1, \{x/b, y/b\}\rangle} \\
&I'_2 = I'_1 \cup \big\{Q(b,z_1,b), Q(z_1,b,b)\big\} \xRightarrow{\theta=\{z_1/b\}} \theta(I'_2) \subseteq I'_1
\end{aligned}
$$

Notice that as recently shown in [75], if the given rule set is single head (i.e., all rules in it are single head), then the fairness for the restricted chase termination is irrelevant. However, unlike the Skolem chase variant for which there is a straightforward termination-preserving translation from any rule set to a single head rule set (cf. [18]), no such termination-preserving translation exists for the restricted chase.

Clearly, given an integer k and a cycle function Φ, any rule set that is $k\text{-}\mathsf{safe}(\Phi)$ under the restricted chase is also $k\text{-}\mathsf{safe}(\Phi)$ under the Datalog-first chase. Example 18 shows that this inclusion relation is indeed strict. The same argument holds for δ-bounded rules under the restricted versus the Datalog-first chase using the same example to demonstrate that the inclusion is strict.

Chapter 6

Distributed Reasoning for Restricted Weakly-Linear Disjunctive Tuple-Generating Dependencies

In this chapter we present our contributions regarding theory and practice of the problem of distributed reasoning over database components for a number of query languages. A condensed version that gives these results is to appear in the International Joint Conference on Rules and Reasoning (RuleML+RR) 2020.

This chapter concerns the problem of querying incomplete data in the presence of an ontology. In this setting, the term ontology-mediated query is attributed to a database query along with an ontology. On aspect critical in answering ontology-mediated queries, specially in scenarios where there are database sources of high volume, is the problem of distributed reasoning which asks whether the query workload can be distributed among different machines, i.e., whether the answer to a query in the presence of an ontology can be computed by parallelizing it over the connected components of the database. If the answer to this problem is positive for a given ontology-mediated query, then we can compute the query in a coordination-free and distributed manner.

Note that the above problem is in general undecidable when the database query is conjunctive and the ontology is defined by a set of Datalog rules [8].

In this chapter, on the theoretical side, we present our contributions on the

problem of distributed reasoning for the class of disjunctive tuple-generating dependencies and we identify a fragment of ontology-mediated queries constructed from this class and conjunctive queries, for which the above problem is decidable.

To the best of our knowledge, this is the first time the problem of distribution over connected database components is studied for this class of dependencies. Though the theory and framework of distributed reasoning with OMQs appeared in [32], to the best of our knowledge, there is no practical evaluation of the theory for any class of OMQs in the literature. To address this issue, we conduct experiments to evaluate the performance gain of reasoning with distributable ontology-mediated queries for the classes of linear as well as linear disjunctive tuple-generating dependencies.

The rest of this chapter is organized as follows. In Section 6.1, the notions and notations used for understanding our contributions of this chapter are introduced. We introduce the class of restricted weakly-linear disjunctive tuple-generating dependencies, which is a new class of rule sets that extends the classes of linear tuple-generating dependencies as well as weakly-linear disjunctive Datalog under a certain syntactic restriction, in Section 6.2.

Then, a class of OMQs, constructed from a subset of the above TGDs, which we identify as bidirectionally-guarded (BG) OMQs is introduced in Section 6.3. This class is interesting in a sense that for which one can establish characterizations of distribution over components that lead to decidable procedures for this purpose. The results of these characterizations are presented in Section 6.4. The procedures introduced in this section lead to 2ExpTime algorithms to decide distribution over components for BG queries.

Then, in Section 6.5, we introduce a syntactic subset of these queries for which the above problem can be done in ExpTime. For this purpose, we utilize UCQ-rewritability of these queries to achieve our goal. Relations to some other fragments are presented in Section 6.6.

Finally, in Section 6.7, we empirically evaluate the performance of distributability checking on real-world ontologies and compare forward chaining-based query answering for centralized and distributed scenarios on these on-

tology benchmarks.

Note that, in this and the next chapter, our theoretical results are obtained mainly by building on existing techniques from the literature. Though the techniques we use are not completely original, the smart application, combination, and adaption for our need allowed us to derive new knowledge, namely some classes of OMQs that support disjunctioin (and the construct of transitive closure given in the next chapter) are indeed distributable and distribution can improve query answering, sometimes substantially.

6.1 Preliminaries

Let C and V be pairwise disjoint countably infinite sets of *constants* and *variables*. A *schema* is a finite set $\mathbf{S}$ of predicate symbols where each symbol $R \in \mathbf{S}$ has an *arity*, denoted $arity(R)$. We introduce special predicate symbol $\top$. *Terms* are elements in $\mathsf{C} \cup \mathsf{V}$. An *atom* over $\mathbf{S}$ is an expression of the form $R(\mathbf{t})$, where $R \in \mathbf{S}$ and $\mathbf{t} \in (\mathsf{C} \cup \mathsf{V})^{arity(R)}$. A *fact* is an atom where terms are constants from C. An *instance* over a schema $\mathbf{S}$ is a set of atoms. A *database* over $\mathbf{S}$ is a finite instance that contains only facts over $\mathbf{S}$. The *active domain* of an instance I, denoted $adom(I)$, is the set of all terms occurring in I.

A *substitution* from a set of symbols U to another set of symbols W is a function $h : U \to W$ defined as: $\varnothing$ is a substitution, and for all $t \in U$ and $t' \in W$, if h is a substitution, then so is $h \cup \{t \to t'\}$.

We write $h(t) = t'$ if $t \to t' \in h$. The *restriction* of h to a subset $D \subseteq U$, denoted $h|_D$, is the substitution $h' = \{t \to h(t) \mid t \in D\}$.

Given two instances I and J (over the same schema), a *homomorphism* $h : I \to J$ is a substitution on terms that is identity on constants and for every atom $R(\mathbf{t})$ of I we have that $R(h(\mathbf{t})) \in J$ which may be alternatively written as $h(R(\mathbf{t}))$ is an atom of J.

A query over $\mathbf{S}$ is a mapping q that maps every database D over $\mathbf{S}$ to a set of *answers* $q(D) \subseteq adom(D)^n$, where $n \geq 0$ is the arity of q. If $n = 0$, then q is a *Boolean query*.

A *conjunctive query* (CQ) q over $\mathbf{S}$ is a formula of the form $q = \exists \mathbf{y}\, \phi(\mathbf{x}, \mathbf{y})$,

where $\mathbf{x}$ and $\mathbf{y}$ are tuples of variables in V and $\phi(\mathbf{x},\mathbf{y})$ is a conjunction of atoms over $\mathbf{S}$ with variables in $\mathbf{x} \cup \mathbf{y}$. A CQ is *answer-guarded* if it has an atom that contains $\mathbf{x}$, and it is *acyclic* if its hypergraph is α-acyclic (cf. [79]). Furthermore, it is *quantifer-free* if $\mathbf{y} = \varnothing$.

The free variables of q are called *answer variables*. The *evaluation* of a CQ q over an instance I, denoted $q(I)$, is defined as the set of all tuples $h(\mathbf{t})$ of constants such that h is a homomorphism from q to I. A *union of conjunctive queries* (UCQ) is a disjunction of CQs that share the same answer variables.

With CQ (resp. UCQ), we denote the class of all queries definable by some CQ (resp. UCQ).[1]

A *disjunctive tuple-generating dependency* (DTGD, also called a *rule*) σ is a first-order (FO) formula $\forall\mathbf{x}\big(\phi(\mathbf{x}) \rightarrow \bigvee_{i=1}^{n} \exists\mathbf{y}_i\, \psi_i(\mathbf{x}_i,\mathbf{y}_i)\big)$, where ϕ and ψ_i for each $i \in \{1,\ldots,n\}$ are conjunctions of atoms, and $\bigcup_{i=1}^{n} \mathbf{x}_i \subseteq \mathbf{x}$ and $\mathbf{x} \cup \bigcup_{i=1}^{n} \mathbf{y}_i \subset \mathsf{V}$. We call $\mathbf{x}$, $\bigcup_{i=1}^{n} \mathbf{x}_i$, and $\bigcup_{i=1}^{n} \mathbf{y}_i$ the set of *universal*, *frontier* and *existential* variables of σ, respectively.

The formula ϕ (resp. $\bigvee_{i=1}^{n} \psi_i$) is called the *body* of σ, denoted $\mathit{body}(\sigma)$ (resp. the *head* of σ, denoted $\mathit{head}(\sigma)$). We define σ to be satisfied by an instance I, written as $I \models \sigma$, if the following condition holds: whenever there exists a homomorphism h such that $h(\phi(\mathbf{x})) \subseteq I$, then there exists a homomorphism $h' \supseteq \{x \rightarrow t \;\in\; h|_{\mathbf{x}} \;\big|\; t \in \mathsf{C} \cup \mathsf{N}\}$, such that at least one disjunct ψ_i of ψ is mapped into I by h'. An instance I satisfies a set Σ of DTGDs, denoted $I \models \Sigma$, whenever for all $\sigma \in \Sigma$, we have $I \models \sigma$.

The set of predicates appearing in Σ is called the *schema* of Σ, denoted $\mathit{sch}(\Sigma)$. For brevity, in the rest of this chapter, we will omit universal quantifiers in front of DTGDs, and instead of $\wedge$, use a comma for conjoining atoms.

A DTGD without any disjunction is called a *tuple-generating dependency* (TGD). Let us denote by TGD the set of all TGDs. A *Datalog rule* is a TGD without existential variables. A finite set of Datalog rules is called a *Datalog*

[1]Let $\mathcal{Q}$ be a class of query language. A query Q is definable by some query $Q' \in \mathcal{Q}$ if Q can be rewritten into $Q' \in \mathcal{Q}$ while preserving the answers of Q to all databases. As an example, consider the query $Q = \exists xy\, A(x,y,z) \vee A(y,x,z) \in \mathsf{UCQ}$. Then, Q is definable by the CQ $Q' = \exists xy\, A(x,y,z) \in \mathsf{CQ}$.

rule set. A *disjunctive Datalog rule* is a DTGD without existential variables. We denote by $\mathsf{Dat}^{\vee}$ the set of all finite disjunctive Datalog rules.

We introduce another special symbol $\square$ as a blank symbol.[2] Each disjunctive Datalog rule set Σ considered in this chapter is assumed to contain the set $\Sigma_{\top}$ which is the union of the following sets $\{Q(x_1, \ldots, x_n) \rightarrow \top(x_1, \square)\}, \ldots,$ $\{Q(x_1, \ldots, x_n) \rightarrow \top(x_n, \square)\}$ for each predicate $Q \in sch(\Sigma)$, where $\top$ is a special predicate of arity 2.[3] We assume that $\square$ only occurs inside $\top$ which does not occur in the head of any rule in Σ.

We say a TGD set Σ' is a *rewriting* of a CQ q w.r.t. a set of DTGDs Σ if there exists a predicate P_q such that for each database D over the schema of Σ, and for each tuple of constants $\mathbf{a}$, we have $D \cup \Sigma \models q(\mathbf{a})$ if and only if $D \cup \Sigma' \models P_q(\mathbf{a})$. A set Σ' of TGDs is a rewriting of Σ if it is a rewriting of every atomic query over $sch(\Sigma)$.

A rule σ is *linear* if it has at most one body atom. A rule set Σ is *linear* if all rules of Σ are linear. The set of all linear TGDs (resp. linear DTGDs) is denoted by L (resp. $\mathsf{L}^{\vee}$). Furthermore, $\mathsf{LDat}^{\vee}$ denotes the set of all linear disjunctive Datalog rules.

An *ontology-mediated query* (OMQ) over $\mathbf{S}$ is a triple $Q = (\mathbf{S}, \Sigma, q)$ in which $\mathbf{S}$ is called the *data schema*, Σ is a finite set of DTGDs and q is a CQ over $\mathbf{S} \cup sch(\Sigma)$. The semantics of Q can be given in terms of certain answers. Given an OMQ $Q = (\mathbf{S}, \Sigma, q)$ and a database D where $arity(q) = n$, we define the *certain answers* to Q over D as: $\mathsf{ans}(D, \Sigma, q) = \{\mathbf{a} \in \mathsf{C}^n \mid D \cup \Sigma \models q(\mathbf{a})\}$, and semantically interpret Q by assigning $Q(D) = \mathsf{ans}(D, \Sigma, q)$ for all databases D. Let D be a database. Each OMQ as defined above can be seen as a query over $\mathbf{S}$ such that for all databases D the evaluation of that query over the *canonical instance* of D and Σ (which can be constructed by the chase algorithm [65]) coincides with the certain answers to Q over D.

Given two queries Q and Q' over $\mathbf{S}$, $Q \subseteq Q'$ if for every $\mathbf{S}$-database D, $Q(D) \subseteq Q'(D)$. Two queries Q and Q' over $\mathbf{S}$ are *equivalent*, denoted $Q \equiv Q'$,

[2]This symbol is introduced to ensure that our setting work for the problem of deciding distribution over components.

[3]These rules are added to support a key transformation that our approach relies on (Definition 52).

if $Q \subseteq Q'$ and $Q' \subseteq Q$. A query language $\mathcal{Q}'$ is at least as expressive as another query language $\mathcal{Q}$, denoted $\mathcal{Q} \preceq \mathcal{Q}'$, if for every $\mathbf{S}$-query $Q \in \mathcal{Q}$, one can find another $\mathbf{S}$-query $Q' \in \mathcal{Q}'$ such that $Q \equiv Q'$. $\mathcal{Q}$ and $\mathcal{Q}'$ are *equi-expressive*, denoted $\mathcal{Q} = \mathcal{Q}'$, if $\mathcal{Q} \preceq \mathcal{Q}'$ and $\mathcal{Q}' \preceq \mathcal{Q}$. For an OMQ $Q = (\mathbf{S}, \Sigma, q)$, if Σ belongs to a class $\mathcal{C}$, we then say that Q belongs to $\mathcal{C}$.

6.1.1 Distribution over Components

From [8, 32], we know that *connectedness* is a key notion for characterizing the distributable fragments of TGDs.

Definition 50. A finite instance I is called *connected* if for all $x, y \in adom(I)$, there exists a sequence $\beta_1, \ldots, \beta_n$ of atoms in I such that a) $x \in adom(\beta_1)$ and $y \in adom(\beta_n)$, and b) for each $1 \leq i < n$, $adom(\beta_i) \cap adom(\beta_{i+1}) \neq \emptyset$. Furthermore, $I' \subseteq I$ is called a *component* of I if I' is connected and for every $\alpha \in I \setminus I'$, $I' \cup \{\alpha\}$ is not connected. The set of all components of such an instance I is denoted $co(I)$.

A TGD σ is *connected* if so is its body, and a TGD set Σ is *connected* if every TGD in Σ has this property.

Let $\mathcal{C}$ be a class of TGDs. We denote by $\mathsf{con}\mathcal{C}$ the set of all rules that belong to $\mathcal{C}$ and that are connected. Similarly, conCQ is the class of all queries definable by some CQ that is connected. Also, conUCQ denotes the class of all queries definable by a union of connected CQs.

Let D be a database over the schema $\mathbf{S}$. A query Q over $\mathbf{S}$ is said to *distribute over components* if $Q(D) = \bigcup_{D' \in co(D)} Q(D')$. For simplicity, we call such a query *distributable*. If a query is distributable, then centralized evaluation of Q over a database D can be equivalently computed by evaluating Q over components of D in a communication-free fashion.

6.2 Restricted Weakly-Linear Disjunctive Tuple-Generating Dependencies

In this section, we introduce the class of restricted weakly-linear disjunctive TGDs and show a technique of rewriting for the preparation of the connect-

edness results later in this chapter

Definition 51. The labelled dependency graph $G_\Sigma = (N, E, \mu)$ of a disjunctive TGD set Σ is the smallest labelled digraph such that:

1. N contains all predicates that occur in Σ;
2. for two nodes $P, Q \in N$, and a rule $\sigma \in \Sigma \setminus \Sigma_\top$ if P and Q occur in $body(\sigma)$ and $head(\sigma)$, respectively, then $\sigma \in \mu(P, Q)$; and
3. $(P, Q) \in E$ whenever $\mu(P, Q)$ is nonempty.

A predicate Q depends on a rule $\sigma \in \Sigma$ if G_Σ has a path which ends in Q and involves an edge labelled with σ. A predicate Q is called *non-disjunctive* if it only depends on non-disjunctive rules, and otherwise, it is disjunctive. An atom is *disjunctive* if its predicate is, and otherwise it is called *non-disjunctive*. A rule set Σ is *weakly-linear* (WL) if each rule in Σ has at most one occurrence of a disjunctive predicate in the body.

For technical reasons, in the rest of this section, we focus on a particular subset of weakly-linear TGDs, which we call *restricted*, that satisfies a syntactic condition as follows. Let Σ be a WL rule set. For each rule $\sigma \in \Sigma \setminus \Sigma_\top$ of the form $\chi \wedge Q(\mathbf{t}) \rightarrow \bigvee_{i=1}^{n} P_i(\mathbf{x}_i)$, in which χ is a conjunction of non-disjunctive atoms and $Q(\mathbf{t})$ is a disjunctive atom, let $\mathbf{t}' = \mathbf{t} \setminus (\bigcup_{i=1}^{n} \mathbf{x}_i \cup var(\chi))$, i.e., $\mathbf{t}'$ is the set of all variables which occur in Q but neither in χ nor in the head of σ. We call Σ *restricted weakly-linear* (RWL), if for all rules $\sigma \in \Sigma \setminus \Sigma_\top$ of the above form, we have $\mathbf{t}' = \varnothing$. The set of all restricted weakly-linear DTGDs is denoted by RWL.

Let us first consider the subset of RWL rules without existential rules. In the following, we write $\mathsf{RWL} \cap \mathsf{Dat}^\vee$ to denote the set of all RWL rules that are disjunctive Datalog. In [97], it was shown that there is a polynomial rewriting from any WL disjunctive Datalog to a set of (non-disjunctive) Datalog rules. We use this transformation for a proper subclass, RWL disjunctive Datalog rule sets, with a slight modification to make it suitable for establishing our results

on distribution over components. In the sequel, when we make a reference to Ξ, we are talking about this particular Datalog rewriting.

Briefly, in this transformation fresh binary predicates are introduced for each disjunctive pair of predicates of the given rule set to prove facts about them. In addition, new rules are introduced to initialize the extension of these fresh predicates to ensure that all head variables also occur in the body of rules. Then, the direction of all rules of the original rule set is flipped by moving all atoms from the head to the body and the other way around.

We now define Ξ transformation for $\mathsf{RWL} \cap \mathsf{Dat}^{\vee}$ rule sets.

Definition 52. Let $\Sigma \in \mathsf{RWL} \cap \mathsf{Dat}^{\vee}$. Denote by $\mathsf{Disj}(\Sigma \setminus \Sigma_{\top})$ the set of all disjunctive predicates occurring in $\Sigma \setminus \Sigma_{\top}$. For each pair $(P, Q) \in \mathsf{Disj}(\Sigma \setminus \Sigma_{\top})^2$, we define a fresh predicate P^Q whose arity is $arity(P) + arity(Q)$. We define $\Xi(\Sigma)$ as the union of rules (1-5) below, where $P_i \in \mathsf{Disj}(\Sigma \setminus \Sigma_{\top})$ for all $1 \leq i \leq n$, where $n \geq 2$; furthermore, $\mathbf{y}$ and $\mathbf{z}$ are vectors of fresh variables such that $\mathbf{y} \cap \mathbf{z} = \emptyset$:

1. a rule $\top(x_1, \square), \ldots, \top(x_n, \square) \rightarrow R^R(\mathbf{y}, \mathbf{y})$ for each $R \in \mathsf{Disj}(\Sigma \setminus \Sigma_{\top})$, where $\mathbf{y} = \{x_1, \ldots, x_n\}$;
2. a rule $\chi \wedge \bigwedge_{i=1}^{n} P_i^R(\mathbf{x}_i, \mathbf{y}) \rightarrow Q^R(\mathbf{t}, \mathbf{y})$ for each rule $\chi \wedge Q(\mathbf{t}) \rightarrow \bigvee_{i=1}^{n} P_i(\mathbf{x}_i) \in \Sigma \setminus \Sigma_{\top}$ and every $R \in \mathsf{Disj}(\Sigma \setminus \Sigma_{\top})$, where χ is a conjunction of Datalog atoms (the same below);
3. a rule $\chi \wedge \bigwedge_{i=1}^{n} P_i^R(\mathbf{x}_i, \mathbf{y}) \rightarrow R(\mathbf{y})$ for each rule $\chi \rightarrow \bigvee_{i=1}^{n} P_i(\mathbf{x}_i) \in \Sigma \setminus \Sigma_{\top}$ and every $R \in \mathsf{Disj}(\Sigma \setminus \Sigma_{\top})$;
4. a rule $Q(\mathbf{z}) \wedge Q^R(\mathbf{z}, \mathbf{y}) \rightarrow R(\mathbf{y})$ for each pair $(Q, R) \in \mathsf{Disj}(\Sigma \setminus \Sigma_{\top})^2$; and
5. each rule σ of Σ with no occurrence of disjunctive predicates.

A predicate of the form P^Q in the transformation means that, given database D and vectors of constants $\mathbf{a}$ and $\mathbf{b}$, if $P^Q(\mathbf{a}, \mathbf{b})$ holds in $D \cup \Xi(\Sigma)$, then proving $P(\mathbf{a})$ suffices for proving $Q(\mathbf{b})$ in $D \cup \Sigma$, i.e., $D \cup \Sigma \models P(\mathbf{a}) \rightarrow Q(\mathbf{b})$. This is explicitly encoded in group 4 of rules.

The group 1 of rules in Definition 52 (to derive facts of the form $R^R(\mathbf{y}, \mathbf{y})$) are used to initialize the extension of auxiliary predicates. The group 2 of rules say that if for all $1 \leq i \leq n$, proving $P_i(\mathbf{x}_i)$ suffices to prove $R(\mathbf{y})$ and χ holds, then proving $Q(\mathbf{t})$ suffices to prove $R(\mathbf{y})$. Since for $R(\mathbf{y})$ to be proved from proving $P_i(\mathbf{x}_i)$ we need to ensure this is the case for every $1 \leq i \leq n$, the disjunction in the given rule is transformed to a conjunction in the transformed rule.[4] Group 3 of rules are defined similarly.

As can be seen, the above transformation is quadratic and in addition, the arity of predicates is doubled at most. The Ξ transformation of Definition 52 is similar to that of Kaminski et al. [97]. The difference of Ξ with their transformation is in the way we have defined $\top$ predicates as compared to theirs: unlike our definition of $\top$ predicates which has arity 2, the arity of $\top$ predicates in [97] is 1.

Theorem 53. *Let Σ be an RWL disjunctive Datalog rule set. $\Xi(\Sigma)$ is a polynomial Datalog rewriting of Σ.*

Proof. Let $\Sigma \in \mathsf{RWL} \cap \mathsf{Dat}^{\vee}$. By construction, $\Xi(\Sigma)$ is a Datalog rule set of size quadratic in the size of Σ. The correctness of Ξ transformation can be shown by induction on hyperresolution derivations of facts which are entailed by the rules in Σ from a given database D. By replacing occurrences of $\top(x)$ in their proof with $\top(x, \square)$ and making proper adjustments, the proof of Theorem 2 of [97] can be directly adopted. □

Example 19 illustrates a RWL rule set and its transformation under Ξ.

Example 19. *Let us consider the following set of Datalog rules $\Sigma = \{\sigma_1, \sigma_2, \sigma_3\}$ in which $\{A, E\}$ and $\{C, B\}$ are sets of Datalog and disjunctive predicates of Σ, respectively.*

$$\sigma_1 : A(x, y) \rightarrow B(x, y) \vee C(x, y)$$
$$\sigma_2 : C(y, z), E(x, y) \rightarrow B(x, z)$$
$$\sigma_3 : B(y, z), E(x, y) \rightarrow C(x, z)$$

[4] Suppose we have $P_1(a) \vee P_2(b)$ in the head of a given rule. Suppose further that in one model M, $P_1(a), R(c) \in M$ and $P_2(b) \notin M$, and in another model M', $P_2(b) \in M'$ and $P_1(a), R(c) \notin M'$, then $R(c)$ cannot be derived even though the given rule is satisfied by both.

It is easy to verify that Σ *is RWL. For such a rule set, in order to prove facts on Datalog predicates, we only rely on the input database* D*. Furthermore,* Ξ *is defined such that to prove facts about disjunctive predicates* C *and* B*, auxiliary fresh predicates of the form* X^Y *are introduced, where* $X, Y \in \{B, C\}$*. The way facts about these disjunctive predicates are proved is as follows: if a fact* $X^Y(a, b, c, d)$ *holds in* $D \cup \Xi(\Sigma)$ *then it suffices to prove* $X(a, b)$ *in order to prove* $Y(c, d)$ *in* $D \cup \Sigma$*.*

In addition, these fresh predicates are initialized by rules of the following form: $\top(x, \square), \top(y, \square) \rightarrow X^X(x, y, x, y)$ *for* $X \in \{B, C\}$*. Then, the rules which involve* B *or* C *are flipped by moving all disjunctive atoms from the head to the body, and the other way around, while their predicates are replaced by auxiliary predicates. Therefore, the transformed rule set will be:*

$$
\Xi(\Sigma) = \begin{array}{l}
\{\sigma'_1 : \top(x, \square), \top(y, \square) \rightarrow B^B(x, y, x, y), \\
\sigma'_2 : \top(x, \square), \top(y, \square) \rightarrow C^C(x, y, x, y), \\
\sigma'_3 : A(x_1, y_1), B^B(x_1, y_1, x_2, y_2), C^B(x_1, y_1, x_2, y_2) \rightarrow B(x_2, y_2), \\
\sigma'_4 : A(x_1, y_1), B^C(x_1, y_1, x_2, y_2), C^C(x_1, y_1, x_2, y_2) \rightarrow C(x_2, y_2), \\
\sigma'_5 : B^B(x_1, z_1, x_2, y_2), E(x_1, y_1) \rightarrow C^B(y_1, z_1, x_2, y_2), \\
\sigma'_6 : B^C(x_1, z_1, x_2, y_2), E(x_1, y_1) \rightarrow C^C(y_1, z_1, x_2, y_2), \\
\sigma'_7 : C^B(x_1, z_1, x_2, y_2), E(x_1, y_1) \rightarrow B^B(y_1, z_1, x_2, y_2), \\
\sigma'_8 : C^C(x_1, z_1, x_2, y_2), E(x_1, y_1) \rightarrow B^C(y_1, z_1, x_2, y_2), \\
\sigma'_9 : B(x_1, y_1), B^B(x_1, y_1, x_2, y_2) \rightarrow B(x_2, y_2), \\
\sigma'_{10} : B(x_1, y_1), B^C(x_1, y_1, x_2, y_2) \rightarrow C(x_2, y_2), \\
\sigma'_{11} : C(x_1, y_1), C^B(x_1, y_1, x_2, y_2) \rightarrow B(x_2, y_2), \\
\sigma'_{12} : B(x_1, y_1), B^C(x_1, y_1, x_2, y_2) \rightarrow C(x_2, y_2)\}
\end{array}
$$

Based on the meaning of fresh predicates of the form X^Y*, e.g., intuitively,* σ'_4 *states that if we can establish that proving* $B(x_1, y_1)$ *and* $C(x_1, y_1)$ *suffices to prove* $C(x_2, y_2)$*, and* $A(x_1, y_1)$ *holds, then* $C(x_2, y_2)$ *must hold. It can be seen that this transformation is quadratic and therefore polynomial in the size of* Σ*.*

Now let us turn our attention to DTGDs. For technical reasons, we consider a syntactically-restricted fragment of restricted weakly-linear DTGDs known as guarded.

Definition 54. (Guarded) Given a set A of atoms, an atom $\alpha \in A$ is called a *guard* w.r.t. A if $adom(\alpha) = adom(A)$. If the set of guard atoms w.r.t. A is

nonempty, then A is called *guarded*. A set Σ of DTGDs is called *guarded* (G) if the body of σ when seen as a set of atoms, for each rule $\sigma \in \Sigma$, is guarded. We denote the set of all G rules by G.

In the rest of this chapter, we focus on disjunctive TGDs under a restricted syntax known as *normal form*.

Definition 55. (Normal form) Let Σ be a set of DTGDs. Let further B and H be atoms, and ϕ and ψ disjunctions of atoms. We say that Σ is in *normal form* if each $\sigma \in \Sigma$ is of the form, $B \to \exists \mathbf{z}\, H$ or $\phi \to \psi$. For notational convenience, we denote the set of rules of the form (1) (resp. (2)) with $\Sigma_\exists$ (resp. $\Sigma_\forall$).

Given a set of DTGDs Σ, it is known that such a normal form transformation θ always exists, which partitions Σ into a set of linear TGDs and a disjunctive Datalog rule set and preserves certain answers for acyclic as well as quantifier-free conjunctive queries (cf. Proposition 2 of [5]). Their result applies to our case of rule sets $\Sigma \in \mathsf{RWL} \cap \mathsf{G}$, as it is easy to see that if Σ is RWL, so is $\theta(\Sigma)$, and if Σ is guarded disjunctive, so is $\theta(\Sigma)$. The reason is that for any rule set $\Sigma \in \mathsf{RWL} \cap \mathsf{G}$, the body of any disjunctive Datalog rule in $\theta(\Sigma)$ is the body of some rule in Σ. Therefore, by definition, these rules are in $\mathsf{RWL} \cap \mathsf{G}$. Moreover, any TGD in $\theta(\Sigma)$ is linear, which by definition makes it belong to $\mathsf{RWL} \cap \mathsf{G}$.

In the sequel, we will assume that all DTGDs are in normal form. Notice that for the certain answer preservation under the normal form transformation, the restriction to acyclic or a quantifier-free conjunctive queries is essential for our techniques to work, and this is the very reason why the related distributability results later in this chapter have these restrictions as well.

Proposition 56. *Let* $\Sigma = \Sigma_\forall \cup \Sigma_\exists \in \mathsf{RWL} \cap \mathsf{G}$. *Then,* $\mathsf{ans}(D, \Sigma_\forall \cup \Sigma_\exists, q) = \mathsf{ans}(D, \Xi(\Sigma_\forall) \cup \Sigma_\exists, q)$, *where* q *is an atomic query.*

Proof. (sketch) Let $\Sigma_\forall \in \mathsf{Dat}^\vee$ be a disjunctive Datalog rule set. From Theorem 53, $\Xi(\Sigma_\forall)$ is a polynomial rewriting of $\Sigma_\forall$.

It follows that for *all* atomic queries q over $\mathbf{S} \cup sch(\Sigma_\forall)$, $\mathsf{ans}(D, \Sigma_\forall, q) = \mathsf{ans}(D, \Xi(\Sigma_\forall), q)$. To show that adding $\Sigma_\exists$ to $\Sigma_\forall$ does not change certain answer semantics, we construct a model D_c of D and $\Sigma_\forall$ known as a *core instance* [5], which intuitively extends D while ensuring satisfaction of $\Sigma_\forall$ only. Since $\Sigma_\forall$ is guarded, D_c can be extended to also satisfy $\Sigma_\exists$, while $\Sigma_\forall$ is satisfied. When D_c is extended, the entailment of facts are preserved. This will ensure satisfaction of D as well as q. For details of this construction, consult [5]. □

Note that by the definition of Datalog rewriting, the Ξ transformation of Definition 52 only preserves fact entailment (or, equivalently, the answers to all atomic queries). However, for arbitrary CQs we may not be able to find query-independent rewritings. In fact as implicitly shown in [109], for an arbitrary disjunctive Datalog rule set Σ which has at least one disjunctive rule, there cannot exist any Datalog rewriting of Σ that preserves answers to all CQs (even acyclic or quantifier-free ones). In spite of this, in the following, we show that by restricting to a fixed query of a certain type, we are able to compute Datalog rewritings. Our rewriting is a special case of Datalog rewriting on what is known as *markable rule sets* introduced in [98].

Next, we show how to extend the result of Proposition 56 to work for acyclic and quantifier-free CQs. For this purpose, regarding acyclic CQs we exploit the fact that any acyclic CQ can be rewritten into a guarded CQ (cf. Corollary 3 of [78]). Moreover, as we will show in Proposition 57, quantifer-free CQs have a particular syntactic restriction which makes it possible to extend the above result.

In the rest of this chapter, w.l.o.g., we assume that any given acyclic query is guarded.

Proposition 57. *Let* $\Sigma = \Sigma_\forall \cup \Sigma_\exists \in \mathsf{RWL} \cap \mathsf{G}$ *and* $q = \exists \mathbf{y} \phi(\mathbf{x}, \mathbf{y})$ *be a CQ.*

(i) If q is acyclic, and involves at most one disjunctive atom from $\Sigma_\forall$, then for all $\mathbf{S}$-databases D, $\mathsf{ans}(D, \Sigma_\forall \cup \Sigma_\exists, q) = \mathsf{ans}\big(D, \Xi(\Sigma_\forall \cup \{\phi(\mathbf{x}, \mathbf{y}) \rightarrow W(\mathbf{x})\}) \cup \Sigma_\exists, W\big)$, where W is a fresh predicate of arity $|\mathbf{x}|$.

(ii) If q is quantifier-free, i.e., $q = \phi(\mathbf{x})$, and involves at most one disjunctive atom from $\Sigma_\forall$, then for all $\mathbf{S}$-databases D, $\mathsf{ans}(D, \Sigma_\forall \cup \Sigma_\exists, q) = \mathsf{ans}\big(D, \Xi(\Sigma_\forall \cup \{\phi(\mathbf{x}) \to W(\mathbf{x})\}) \cup \Sigma_\exists, W\big)$, where W is a fresh predicate of arity $|\mathbf{x}|$.

Furthermore, for either (i) or (ii), Ξ produces a rule set, as its output, that is polynomial in the size of its input.

Proof. Let $\Sigma = \Sigma_\forall \cup \Sigma_\exists \in \mathsf{RWL} \cap \mathsf{G}$ and W be a fresh predicate. We first prove (i). Since q is acyclic, by our previous assumption, q is guarded. In addition, since q consists of at most one disjunctive atom from $\Sigma_\forall$, then, $\Sigma_\forall \cup \{\phi(\mathbf{x}, \mathbf{y}) \to W(\mathbf{x})\}) \cup \Sigma_\exists$ is in $\mathsf{RWL} \cap \mathsf{G}$. The rest of the proof follows from Proposition 56. In order to prove (ii), we utilize the fact that the syntactic restriction of quantifier-free queries ensures that the required variable assignments for answering queries are such that they only map into the constants of the input database. This fact combined with Proposition 56 leads to the desired conclusion. □

6.3 Bidirectionally-Guarded Queries

In this section we introduce the class of *bidirectionally-guarded queries*: A new class of queries that consists of rule sets that extend subclasses of existential rules as well as linear disjunctive Datalog rules.

We consider the class of OMQs where a query q involves at most one disjunctive atom from the underlying $\Sigma_\forall$. Let us denote this class by $(\mathsf{RWL} \cap \mathsf{G}, \mathsf{Q})$, where Q is the class of queries definable by answer-guarded CQs that are either acyclic or quantifier-free. In general, it is unknown whether the problem of distribution over components for this class of OMQs is decidable. In this chapter, we show that we can characterize and decide the problem of distribution over components for a subclass which we call bidirectionally-guarded. Intuitively, decidability can be achieved by imposing the guardedness condition for the transformed rule set under the Ξ transformation.

Definition 58. Let $Q = (\mathbf{S}, \Sigma = \Sigma_\forall \cup \Sigma_\exists, q) \in (\mathsf{RWL} \cap \mathsf{G}, \mathsf{Q})$ in which $q = \exists \mathbf{y} \phi(\mathbf{x}, \mathbf{y}) \in \mathsf{Q}$. Q is called *bidirectionally-guarded* if $\Xi(\Sigma_\forall)$ is guarded. We denote the set of bidirectionally-guarded queries by BG.

The following lemma can be verified using the definition of the Ξ transformation.

Lemma 59. *A query $Q = (\mathbf{S}, \Sigma, q) \in (\mathsf{RWL} \cap \mathsf{G}, \mathsf{Q})$ is bidirectionally-guarded if it satisfies the following conditions:*

1. *for each rule $\chi \wedge Q(\mathbf{t}) \rightarrow \bigvee_{i=1}^{n} P_i(\mathbf{x}_i) \in \Sigma \setminus \Sigma_\top$, we have: (i) $var(\chi) \subseteq \bigcup_{i=1}^{n} \mathbf{x}_i$; and (ii) $\bigwedge_{i=1}^{n} P_i(\mathbf{x}_i)$ has a guard atom, i.e., $\exists\, i$ s.t., $var(P_i(\mathbf{x}_i)) = var(\bigwedge_{i=1}^{n} P_i(\mathbf{x}_i))$;*
2. *for each rule $\chi \rightarrow \bigvee_{i=1}^{n} P_i(\mathbf{x}_i) \in \Sigma \setminus \Sigma_\top$, we have: (a) $var(\chi) = \bigcup_{i=1}^{n} \mathbf{x}_i$; and (b) $\bigwedge_{i=1}^{n} P_i(\mathbf{x}_i)$ has a guard atom, i.e., $\exists\, i$ s.t., $var(P_i(\mathbf{x}_i)) = var(\bigwedge_{i=1}^{n} P_i(\mathbf{x}_i))$.*
3. *The maximum arity of all disjunctive predicates occurring in $\Sigma \setminus \Sigma_\top$ is 1.*

Note that the restriction imposed on the arity of disjunctive predicates is to ensure that the output of Ξ on $\Sigma_\forall$ is guarded. Moreover, the requirement that the rule sets in BG queries are guarded guarantee that the certain answers are preserved.[5] This condition is needed in establishing the main results of this chapter.

Example 20. *Consider the rule set of Σ_1 of the introduction section. It can be verified that any query $Q = (\mathbf{S}, \Sigma, q) \in (\mathsf{RWL} \cap \mathsf{G}, \mathsf{Q})$ is bidirectionally-guarded. where $\Sigma = \{\sigma_1, \ldots, \sigma_7\}$.*

Based on Definition 58, it is clear that $\mathsf{BG} \preceq (\mathsf{RWL} \cap \mathsf{G}, \mathsf{Q})$. In what follows, we study the problem of membership checking in BG queries.

Proposition 60. *Given a query $Q = (\mathbf{S}, \Sigma, q) \in (\mathsf{RWL} \cap \mathsf{G}, \mathsf{CQ})$, the membership problem of Q in BG is in* PTime.

[5]Note that guardedness here refers to the given DTGDs.

Proof. We need to do the following: (i) checking whether q has at most one disjunctive atom, (ii) applying Ξ transformation on $\Sigma_\forall \cup \{\phi(\mathbf{x},\mathbf{y}) \rightarrow W(\mathbf{x})\}$, and (iii) checking whether $\Xi(\Sigma_\forall \cup \{\phi(\mathbf{x},\mathbf{y}) \rightarrow W(\mathbf{x})\})$ is guarded. It is easy to see that (i) can be done within a PTIME upper bound. According to Proposition 57, task (ii) can be done in PTIME. In addition, from the conditions of Lemma 59, it is easy to verify that checking (iii) can be done in PTIME as well.

Then, we need to see if the given query $q \in \mathsf{CQ}$ is answer-guarded which can be done in LOGSPACE. Moreover, we construct the hypergraph of q and check whether it is acyclic. The latter two tasks can be done in SL (Symmetric LOGSPACE)[6] [79], which based on [125] is equal to LOGSPACE.

The same statement of above also holds true for the answer-guarded qunatifier-free CQs. Therefore the overall complexity upper bound of checking membership of Q in BG is in PTIME. □

6.4 The Problem of Distribution over Components for Bidirectionally-Guarded Queries

A key property of the rule sets of BG queries is that they are connected since they are guarded. Another useful property of BG queries is presented in Lemma 61 below which states that the problem of containment checking for these queries is decidable.

Lemma 61. *Given two queries $Q_1, Q_2 \in \mathsf{BG}$, the problem of checking whether $Q_1 \subseteq Q_2$ is decidable, and is in* 2EXPTIME.

Proof. Given two queries $Q_1, Q_2 \in \mathsf{BG}$, as the rule sets of Q_1 and Q_2 are guarded TGDs, the problem of checking whether $Q_1 \subseteq Q_2$ is an instance of the problem of checking the containment of guarded TGDs with answer-guarded (acyclic or quantifier-free) queries. Such queries can be naturally translated into GNFO

[6]Recall that SL is the complexity class of problems which are logspace reducible to *undirected s-t connectivity*, that is the problem of determining whether there exists a path between two vertices in an undirected graph.

OMQs [21], and the containment problem for GNFO OMQs is known to be in 2ExpTime [21] which gives an upper bound for the desired problem. □

Now, we get back to the problem of distribution analysis over components for $(\mathsf{RWL} \cap \mathsf{G}, \mathsf{CQ})$ queries. Let DIST be the class of queries that distributes over components. We can show

Theorem 62. $\mathsf{BG} \cap \mathsf{DIST} = \mathsf{conBG}$.

Proof. The proof proceeds by establishing the following results: (1) $\mathsf{BG} \cap \mathsf{DIST} \preceq \mathsf{conBG}$; (2) For $\mathcal{Q} = (\mathsf{RWL} \cap \mathsf{G}, \mathsf{Q})$, $\mathsf{con}\mathcal{Q} \preceq \mathcal{Q} \cap \mathsf{DIST}$; and (3) $\mathsf{BG} \preceq (\mathsf{TGD}, \mathsf{CQ})$. For (1), we fix an arbitrary query $Q = (\mathbf{S}, \Sigma, q) \in \mathsf{BG}$, where q is defined by $\exists \mathbf{y} \phi(\mathbf{x}, \mathbf{y})$. Let $A = \{\phi_1, \ldots, \phi_k\}$ be the set of connected components of q, and q_i be the CQ that is obtained from q by keeping only the component ϕ_i. Since $Q_i = (\mathbf{S}, \Sigma, q_i) \in \mathsf{conBG}$, it is sufficient to prove that Q_i is equivalent to Q. This proof utilizes the fact that Σ is connected. Furthermore, evaluating equivalence of queries involves containment checking which is proved for BG in Lemma 61. Claim (2) is already proved in [32] (which even works after relaxing the answer-guardedness requirement of queries), and (3) is trivial. □

We will now consider a subset of BG queries, denoted BG^S. A query $Q = (\mathbf{S}, \Sigma = \Sigma_\forall \cup \Sigma_\exists, q) \in \mathsf{BG}$ is in BG^S, called *singly bidirectionally-guarded* (hence, S in the superscript of the class notation) if $\Sigma_\forall \in \mathsf{L}$.

There are two reasons why we consider this fragment: (1) in the next section we show that by focusing on this syntactic restriction, we can substantially reduce the complexity of distribution checking compared to that of BG; and (2) in our experiments we use this class of queries.

Example 21. *Let $\Sigma_1 = \{\sigma_1, \sigma_2, \sigma_3, \sigma_4\}$ be the first four rules of the rule set of the introduction section. Then, each query $Q = (\mathbf{S}, \Sigma_1, q)$ for $q \in \mathsf{Q}$ belongs to BG^S. This can be verified by applying the transformation Ξ.*

As a corollary to Theorem 62, we have:

Corollary 63. $\mathsf{BG}^S \cap \mathsf{DIST} = \mathsf{conBG}^S$.

To summarize this section, $\mathsf{con}(\mathsf{RWL} \cap \mathsf{G}, \mathsf{Q})$ provides an effective syntax for $(\mathsf{RWL} \cap \mathsf{G}, \mathsf{Q})$ queries that distribute over components, while the problem of deciding whether such a query distributes over components may be undecidable. As a sufficient condition to guarantee decidability, we introduced a subclass BG of these queries that guarantee decidability of the above problem using a decision procedure on the membership of distributable queries as sketched in the proof of Theorem 62. Then, to tame the high complexity of this problem, we introduced a subclass BG^S of BG queries.

6.5 Deciding Distributability via Rewriting

In Theorem 62 and Corollary 63, we provided semantic characterizations for checking membership in DIST for queries consisting of BG and BG^S queries, respectively. Recall that these queries are subsets of $(\mathsf{RWL}, \mathsf{Q})$. Our semantic characterization was based on decidability of the containment problem for the latter queries. Therefore, the procedure provided in the proof of Theorem 62, which is based on Lemma 61, runs in 2ExpTime, already provides a decision algorithm to decide membership of these queries in DIST.

In this section we show that this result can be strengthened to provide decision procedures for the above problem in the case of BG^S queries while reducing the complexity upper bound for membership checking to single exponential time.

The key tool in establishing distributability results for BG^S queries is what is known as UCQ rewritability which was originally introduced in the context of description logics [48]. This property guarantees high tractability of data complexity and makes the reasoning feasible by relational query engines as the evaluation of UCQ queries is in the (highly tractable) class AC_0 in data complexity [141]. In more details, if $Q = (\mathbf{S}, \Sigma, q(\mathbf{x}))$ is an OMQ, then Q is *UCQ-rewritable* if there exists a (finite) UCQ $Q_\Sigma(\mathbf{x})$ over $\mathbf{S}$, known as the *perfect rewriting* [80], that is *equivalent* to Q, i.e., for every $\mathbf{S}$-database D, we have $Q(D) = Q_\Sigma(D)$. In this case, $Q_\Sigma(\mathbf{x})$ is called a UCQ-rewriting of Q. The notion of UCQ-rewritability of an OMQ naturally extends to a class of OMQs.

In this section, we show how to utilize UCQ-rewritability of OMQs composed of linear disjunctive TGDs with conjunctive queries, as reported in [38], to characterize distribution over components for BG^S queries. Throughout this section, we will exploit a perfect rewriting algorithm for our purpose. Roughly speaking, given a rule set Σ and a conjunctive query $q(\mathbf{x})$, a perfect rewriting algorithm rewrites $q(\mathbf{x})$ into a UCQ, called *UCQ rewriting*, of minimal cardinality (i.e., the number of conjunctive queries in the resulting UCQ is minimum), using the rules in Σ such that for any $\mathbf{S}$-database D, the answers to $q(\mathbf{x})$ over the OMQ $(\mathbf{S}, \Sigma, q(\mathbf{x}))$ can be computed by evaluating the rewritten query directly over D.

Given an OMQ Q, let $\mathsf{Dist}(Q)$ denote the decision problem of distribution over components for Q. Our rewriting-based checking mechanism to decide $\mathsf{Dist}(Q)$, where $Q \in \mathsf{BG}^S$, is presented in Algorithm 6.1. We will use the following notations. Let $Q = (\mathbf{S}, \Sigma, q(\mathbf{x})) \in \mathsf{BG}^S$. Denote by $RW(Q)$ the UCQ rewriting result of Q using some perfect rewriting algorithm RW. To proceed, as discussed above, for each answer-guarded (acyclic or quantifier-free) CQ q, we can rewrite q using any given Σ into a UCQ, or in other words, construct $RW(Q)$ and then continue with the approach described in Algorithm 6.1.

Algorithm 6.1 takes as input a query $Q \in \mathsf{BG}^S$ which consists of a rule set Σ and an answer-guarded (acyclic or quantifier-free) conjunctive query q, and returns true if Q is distributable and false, otherwise. It computes a UCQ-rewriting Q' of Q. Once a UCQ-rewriting Q' of a given query Q is computed, it is clear that $Q \in \mathsf{DIST}$ if and only if $Q' \in \mathsf{DIST}$. In the rest of the algorithm, a mechanism to decide $\mathsf{Dist}(Q')$ is presented. However, as observed in [32], we cannot proceed by just checking if disjuncts of Q' distribute over components. The reason is that some disjuncts which are not distributable may be subsumed by the others which are distributable. Lemma 64 was proposed in [32] as a solution to the above problem.

Let us define a mapping $e : \mathsf{V} \cup \mathsf{C} \to \langle \mathsf{V}, * \rangle \cup \mathsf{C}$, where constants in C are mapped to themselves and each variable $v \in \mathsf{V}$ is mapped to $\langle v, * \rangle$, where $\langle v, * \rangle$ is a fresh constant constructed from v. Given a conjunctive query $q(\mathbf{x}) = \exists \mathbf{y}(\bigwedge_{i=1}^{n} R_i(\mathbf{x}, \mathbf{y}))$, let $D[q]$ be the transformation of q after applying mapping

e to q. Therefore, we define $D[q] = e(q(\mathbf{x})) = \bigwedge_{i=1}^{n} R_i(\langle \mathbf{x}, * \rangle, \langle \mathbf{y}, * \rangle)$. Since each query $Q \in \mathsf{BG}^S$ can be rewritten into a UCQ, then the following lemma applies to these queries and leads to a concrete decision procedure for membership of these queries in DIST.

Lemma 64. *For $Q(\boldsymbol{x}) \in \mathsf{UCQ}$, $Q \in \mathsf{DIST}$ if and only if for each $q \in Q$, there exists $D' \in co(D[q])$ such that $\langle \boldsymbol{x}, * \rangle \in Q(D')$.*

Algorithm 6.1 Checking membership of $Q \in \mathsf{BG}^S$ in DIST

Input: An OMQ $Q \in \mathcal{Q} = \mathsf{BG}^S$;
Output: Boolean value IsDistributable;

```
1: procedure Dist(Q), WHERE Q ∈ 𝒬
2: end procedure
3: IsDistributable ← true; Temp ← false;
4: Construct the UCQ Q'(x) ← RW(Q);
5: for each CQ q' ∈ Q'(x)
6:     for each D' ∈ co(D[q'])
7:         if ⟨x, *⟩ ∈ Q(D')
8:             Temp ← true; break;
9:     if Temp == false
10:        IsDistributable ← false; break;
11: return IsDistributable
```

It is not hard to see that the procedure $\mathsf{Dist}(Q)$, for each $Q \in \mathcal{Q}$, always terminates, since for all the given OMQs the size of rewritten UCQ is always finite [13, 38, 96].

To conclude this section, let us present the complexity results for membership checking in DIST for OMQs consisting of the above introduced fragments of $(\mathsf{RWL} \cap \mathsf{G}, \mathsf{Q})$.

Theorem 65. $\mathsf{Dist}(\mathsf{BG}^S)$ *is in* ExpTime. *Furthermore,* $\mathsf{Dist}(\mathsf{BG})$ *is in* 2ExpTime.

Proof. The combined complexity of evaluation of $\mathsf{L}^\vee$ rules, which is known to be in ExpTime [82], combined with Lemma 64, gives us the desired upper bound for BG^S queries. Regarding $\mathsf{Dist}(\mathsf{BG})$ queries, refer to the procedure detailed in the proof of Theorem 62. □

6.6 Relation to other Formalisms

In [8], it was shown that the problem of deciding whether a Datalog query distributes over components is undecidable. Furthermore, the fragment of Datalog combined with conjunctive queries in which all rule bodies are connected, characterizes all those queries that distribute over components. Additionally, in [32], this problem was considered for ontology-mediated queries with classes of finite sets of TGDs (TGD) as well as linear (L), (weakly) guarded ($(\mathsf{W})\mathsf{G}$), (weakly) sticky ($(\mathsf{W})\mathsf{S}$) sets of TGDs and conjunctive queries. It was shown that the fragment of OMQs $\mathcal{Q} = (\mathcal{C}, \mathsf{CQ})$ that are distributable over components, in which $\mathcal{C}$ belongs to the above-mentioned group of TGDs is precisely the one in which $\mathcal{C}$ and CQ are connected.

In addition, when $\mathcal{C}$ belongs to the set: $\{\mathsf{L}, \mathsf{S}\}$ (resp. $\{\mathsf{G}\}$), then $\mathsf{Dist}((\mathcal{C}, \mathsf{CQ}))$ (resp. $\mathsf{Dist}((\mathcal{C}, \mathsf{AGCQ}))$) is decidable where AGCQ denotes the class of all queries definable by some answer-guarded CQ.

Guarded OMQs Note that guarded OMQs are not UCQ-rewritable, and thus, do not belong to the FUS sets of ontology languages. On the other hand, to the best of our knowledge, the decidability of $\mathsf{Dist}((\mathsf{G}, \mathsf{CQ}))$ is still open. However, more recently, in [23], it was shown that there is a UCQ-rewritable fragment within $(\mathsf{G}, \mathsf{CQ})$ and its membership decision problem is 2ExpTime-complete. So, this leads to decidability of Dist for this fragment and additionally, one can use Algorithm 6.1 for checking membership in DIST for it. Finally, by definition, $(\mathsf{G}, \mathsf{AQ})$ queries are all distributable over components, where AQ is the class of all queries definable by some atomic query.

Frontier-Guarded (FG) OMQs FG TGDs extend guarded fragment by requiring each TGD σ to have at least one guard atom that contains all frontier variables (rather than all universally-quantified variables) of σ. Unlike guarded TGDs, frontier-guarded TGDs are not connected in general. So, Theorem 62 does not hold for FG TGDs, even with $\mathsf{Q} = \mathsf{AQ}$. However, from [23] and [32], we have the following corollary.

Corollary 66. *We have* (i) $(\mathsf{FG}, \mathsf{CQ}) \cap \mathsf{DIST} = (\mathsf{conFG}, \mathsf{conCQ})$*, and* (ii) *for* $Q \in (\mathsf{FG}^{\triangle}, \mathsf{conCQ})$, $\mathsf{Dist}(Q)$ *is decidable, in which* $\mathsf{FG}^{\triangle} \subset \mathsf{FG}$ *is the fragment of*

FG *which is UCQ-rewritable.*

DL-Lite$_{\mathsf{bool}}$ OMQs Fact entailment w.r.t. data for DL-Lite$_{\mathsf{bool}}$ logics are shown to be tractable in [11]. It is known that certain DL-Lite$_{\mathsf{bool}}$ logics can be rewritten as linear DTGDs. So, our results on distributability of distribution over components for BG^S queries can be applied to these logics as well.

6.7 Experiments on OMQs Based on Linear Disjunctive TGDs

The experiments of this section are designed for two purposes. First, after the publication of the theoretical work of distribution [32], there has been no practical evaluation whether query answering can be improved by distribution. For this, we select some ontologies that can be translated into linear rules and conduct experiments. The second objective is to evaluate practical relevance of our theoretical results for disjunctive rules. We test both decidability checking and answering queries.

We conducted two experiments. The first one concerns the evaluation of our distributability checking algorithm and in the second, we evaluate the performance of forward chaining for distributed versus centralized schemes. Both experiments were done on three different ontology benchmarks.

A master-slave architecture was adopted where each slave machine runs a chase engine (for the second experiment) and the partial query results are aggregated back in the master machine to answer the query posed by any client (the first experiment). All experiments were done on a private cluster and on the Amazon EC2 platform. For this purpose, we utilized EC2 instances of type a1.2xlarge. The physical cluster is managed with VMware Fusion version 11.5.3 and a virtual cluster of 8 Virtual Machines (VMs) is provisioned to run the experiments. Each VM is located on a separate physical host and configured with 4 vCPUs, 8GB of RAM and 128GB of local disk. The software on each VM is 64-bit macOS Catalina.

For the second experiment, we compared different statistics derived from our distributed approach, using chase engines RDFox [119] and Graal [17],

against centralized approaches for query processing.

Experimented Ontologies: The first benchmark in our experiments is $\text{LUBM}^{\exists}_{20}$ [110], for which we generated instances with 100K, 500K, and 1M facts with the data generator and singled out axioms that correspond to linear rules.

All manually curated queries are CQs in SPARQL 1.0 syntax. We added 30 more handcrafted queries to the available query pool. Thus, forming 50 overall OMQs up for evaluation.

The second benchmark concerns linear rules from Open Biomedical Ontology (OBO) corpus [134]. For this benchmark, we ended up with 50 terminating linear ontologies for which we handcrafted acyclic CQs and created an initial database for each.

The last benchmark is MOWLCorp corpus [115], which was selected for evaluations on linear disjunctive rules. For the last two corpora, our selected ontologies were those for which the number of their existential axioms was 10. This gives us 41 and 73 linear ontologies from a total of 125 and 132 ontologies from OBO and MOWLCorp corpora, respectively.

For each considered ontology, we perform standard transformation to extract the corresponding DTGDs.[7] For each OMQ $Q = (\mathbf{S}, \Sigma, q)$ as constructed above and targeted query q, we performed the following tasks: (i) acyclicity checking of Σ and q; (ii) membership checking about whether it belongs to BG^S, (iii) checking whether it is distributable, and (iv) distributed reasoning with it.

For testing the acyclicity conditions considered in this chapter for the rule sets, following [38], we replace all occurrences of $\vee$ with $\wedge$, and then check rules for membership in WA [65] and aGRD [12].

To check whether Q belongs to BG^S (for the MOWLCorp corpus), we first find the normalized form for the given rule set (Σ) as described in Definition 55. This can be done by introducing fresh predicates. Then, we apply our implementation of transformation Ξ of Definition 52 on the resulting rule set.

[7] We refer to [89] for details on this standard normalization procedure. Our normalized forms follow their Table 1, but with an additional form $A_1 \sqcap \cdots \sqcap A_n \sqsubseteq B_1 \sqcup \cdots \sqcup B_m$.

Ontology	# Total OMQs	# Dist. OMQs	Avg. dist. checking time (s)	Avg. query rewriting time (s)
$\text{LUBM}^{\exists}_{20}$	50	37	31.5	25.3
OBO	50	42	56.6	43.3
MOWLCorp	100	40	63.2	29.9

Table 6.1: Statistical results for distributability membership checking

Finally we check the conditions for BG^S, manually, on the transformed OMQ. We implemented a module to track dependencies of Σ to output disjunctive atoms. Also, while ensuring all CQs involve at most one occurrence of these atoms, we check acyclicity of CQs manually or handcraft them to be acyclic.

For (iii), since at the time of writing this dissertation, we could not find any implemented approach that can handle rewriting DTGDs, we perform UCQ-rewriting of $\Xi(\Sigma)$ instead. Then, we apply Algorithm 6.1 on Q as a decision module for distributability checking on the master machine for both BG^S queries as well as those consisting of only linear TGDs. We utilize the piece-based query rewriting technique known as *pure* (*piece unification based rewriting*) [102] which outputs a sound and complete UCQ-rewriting of both query types as mentioned above.

For (iv), for each distributable OMQ $Q = (\mathbf{S}, \Sigma, q)$ from step (iii), we perform the chase performance experiments on $\Xi(\Sigma)$, the results of which are depicted in Table 6.2.

After preprocessing and acyclicity checking, for the collection of ontologies from the MOWLCorp, 10 were found to be terminating under tested acyclicity conditions, for each of which we handcrafted 10 acyclic CQs and an initial database for each OMQ was considered. Thus, overall, 100 OMQs were tested from the last corpus.

Table 6.1 shows statistics on distributability checking of the considered corpora. Overall, 50 linear OMQs from the $\text{LUBM}^{\exists}_{20}$ benchmark suite have been tested for distributability and 37 (74%) of them satisfied distributability condition of Algorithm 6.1 and the other 13 did not. For the OBO ontology benchmark, the same number were considered for distributability checking and

Table 6.2: Statistics of distributed versus centralized chase schemes

Ontologies	**Centr.**				**Distr.**				**Avg. # components**
	RDFox		**Graal**		**RDFox**		**Graal**		
	Restr.	Sk.	Restr.	Sk.	Restr.	Sk.	Restr.	Sk.	
$\text{LUBM}^{\exists}_{20}$	65.2	72.3	2375.8	2450.2	24.9	26.7	338.0	343.1	8
OBO	261.4	264.3	4223.0	4264.4	75.1	77.8	904.9	903.4	5
MOWLCorp	342.4	351.6	4237.0	4243.9	58.2	63.7	631.5	640.0	4

42 (84%) satisfied this condition. Furthermore, among 100 OMQs considered from the MOWLCorp corpus, we found 60 (60%) that belong to BG^S. Of these OMQs, 40 (67%) passed the distributability test of Algorithm 6.1 which form 40% of the total number of considered OMQs for this corpus.

The second experiment is conducted over distributable queries. Given a database D which is the result of transforming the input RDF store, the number of components of D for each $Q \in \mathsf{DIST}$ gives us the number of cluster machines needed for our next evaluation.

Node i of the cluster network corresponds to each component D_i of the database D. Then at each node i, we deployed Graal and RDFox chase engines to compute $chase(D_i, R)$ (local materialization) in parallel, considering both the Skolem and the restricted variants of chase for each engine. Then all the results of local computations will be sent and aggregated at the master node. Table 6.2 gives the statistics related to this operation (including comparisons of the chase performance for centralized and distributed schemes as described above) for these two chase engines regarding the evaluation time on the master node for the considered OMQs $Q \in \mathsf{DIST}$. All the averages in the table that represent numbers for which the domain is the set of integers are rounded down in Table 6.2. Note that all computation times are based on the average measure in seconds.

Chapter 7

Distributed Reasoning for Generalized Regular Queries

In this chapter, we study the problem of distributed reasoning as introduced in the previous chapter for a class of queries known as *generalized regular (ontology-mediated) queries* (GRQs), where rules are Datalog rules in which no recursion is allowed except for the rules that express transitive closure. The language of GRQs was shown to be the most expressive fragment of Datalog with a decidable problem of containment checking with which one can express transitive closure constructs [139].

We prove that the distribution problem for the language of GRQs is decidable with the complexity of distribution checking that is upper bounded by 2ExpSpace. In order to reduce this complexity, we introduce a subclass of these queries that we call *k-pseudo-guarded generalized regular queries* (PG2RQ^k), where k is a fixed integer, for which we prove an ExpSpace upper bound for the complexity of distribution checking.We further extend this result to different classes of queries defined in the literature. Finally, we conduct experiments to verify our theoretical results on real-world ontology benchmarks.

This chapter is organized as follows. In Section 7.1, we introduce the necessary mathematical notions and terms required to grasp this chapter. Section 7.2 defines the class of GRQs. The connected fragment of GRQs provides an effective syntax for these queries that are distributable over components. Then, we introduce the class of PG2RQ^k queries.

Section 7.3 is devoted to characterizations for distribution over components

of GRQ and PG2RQ^k queries. Using these characterizations, in Section 7.4, we introduce concrete algorithms for checking distributability of GRQs and PG2RQ^k queries.

Then, in Section 7.5, we show how the approach provided in Section 7.4 can help us establish decidable characterizations for checking distribution over components on a number of fragments introduced in the literature. This section is divided into two subsections. The first subsection studies distribution over components for these fragments without involving transitivity. In the second subsection, transitivity axioms are combined with the fragments introduced in the first subsection, and then we show how distribution over components is affected by this combination.

A case study for a real-world scenario which falls into the class of PG2RQs is introduced in Section 7.6. Finally, in Section 7.7, we report the results of our experiments for checking distribution over components as well as query answering performance for distributable PG2RQ^k queries for a fixed value of k which is reported in that section.

7.1 Preliminaries

For the preliminaries of this chapter, we refer the reader to those of the previous chapter which is complemented with the following notion. Let Σ be a Datalog. The head predicates of Σ are called *intensional* (IDB) and all other predicates of Σ are called *extensional* (EDB); an atom is called *intensional* (IDB) (resp. *extensional* (EDB)), if so is its predicate.

The main distributivity results are obtained by applying existing constructions as well as some novel ones by the author of the thesis.

7.2 Generalized Regular Query Languages

Modern graph query languages are gaining increasing attention due to their usefulness in modelling, navigating and reasoning with interconnected data which is ubiquitous in different applications including, but not limited to, semantic web, social network analysis, ontology-mediated query answering, etc.

Note that there is no single standard language for querying graph databases. In fact, *regular path query language* (RPQ) [67] is the basic language for graph databases [1, 3, 39] which asks for paths given by a regular expression over a given alphabet. This allows a controlled form of recursion to be expressed over binary predicates.

RPQs allow for navigation of the edges of a graph database only in the forward direction. When RPQs are enriched to have the ability to navigate the edges of graph databases in a backward direction as well, the resulting queries are called *two-way regular path queries* (2RPQs) [47].

The language of *conjunctive* 2RPQs (C2RPQs) [47] is introduced as an analogue to CQs for graph databases. A C2RPQ is a conjunctive query in which atoms are the form $P(x, y)$ where P is a 2RPQ. Also similar to union of conjunctive queries or UCQs which extend CQs by incorporating the union operation, UC2RPQs are the class of unions of C2RPQs. On the practical side, the shift is happening from the class of conjunctive queries (CQs) as the fundamental language which corresponds to atomic relational queries closed under selection, conjunction, and projection to more expressive ones. In particular, the core query languages in SPARQL 1.1 which is a more recent W3C standard for querying RDF data has been updated to include C2RPQs. It is known that C2RPQs extend both CQs and RPQs.

It is worthy to note that UC2RPQs extend UCQs, and are therefore closed under the operations of union, selection, projection, and conjunction, but they are not closed under the *transitive closure* (TC) operation. In fact, TC only occurs inside 2RPQs. Therefore, many natural queries cannot be expressed in UC2RPQs. As an example, the query $P(x, y), P(y, z), P(z, x) \rightarrow Q(x, y)$ on a graph database G, where P represents edges of G, asks whether there is a triangle in G. This query belongs to C2RPQs. However, the transitive closure Q^+ of Q does not belong to UC2RPQ. Ideally, we are looking for a query language which (1) is closed under certain operations useful in navigating graph databases including TC; (2) allows for querying over hypergraphs like RDF fact bases or sets of facts with an arbitrary predicate arity; and (3) is tractable.

For this purpose, the class of generalized regular queries is introduced in [126] which satisfies all the above properties of a query language for navigating (hyper)graph databases. This language can be defined as UC2RPQs which are closed under TC.

Although GRQs do not allow to express full recursion, to the best of our knowledge, generalized regular queries are the most expressive fragment of first-order logic with transitive closure which is known to have an elementary containment problem [126]. Furthermore, they offer additional properties over (full) Datalog, some of which are summarized as follows: (1) The complexity of evaluation of GRQ queries is NLogSpace-complete which puts them in the NC class which consists of *highly parallelizable* problems; (2) query containment checking of GRQ queries is decidable with a tight bound, and belongs to 2ExpSpace-complete [126].

The importance of the language of GRQs motivites the question of whether the corresponding queries are distributable and if yes, whether distribution generates practical gains in query answering, which we will address in this chapter.

7.2.1 Pseudo-Guarded Generalized Regular Queries

In this section we present the theory of generalized regular queries, and furthermore, we introduce a novel class of queries which we call *k-pseudo-guarded generalized regular* that is a *syntactically connected* subset of GRQs, and then we study the theory of these queries.

Previous work [8, 32] show positive results on checking distribution over components for guarded, linear and sticky rules, and negative results for Datalog rules. However, none of the abovementioned sets of rules is able to express transitive closure (TC) which is a basic construct in e.g., graph database applications.

The class of generalized regular queries allows us to incorporate transitive closure while ensuring that distributability checking remains decidable. The idea is to have an expressive yet distributable fragment of Datalog by (1) limiting recursion in a way that it is used only to express TC; and (2) restricting

body of rules to guarantee their connectedness.

Before we present our rule fragments let us introduce some notions. Let Σ be a Datalog rule set over schema $\mathbf{S}$. A predicate $P \in \mathbf{S}$ depends on a predicate $Q \in \mathbf{S}$ if Q occurs in the body of some rule $\sigma \in \Sigma$ and P is the predicate that occurs in the head of σ. Furthermore, the dependency graph $G = (N, E)$ of Σ is a directed graph in which N is the set of predicates of Σ and there is an edge $e \in E$ from Q to P if P depends on Q. A rule set Σ is nonrecursive if the dependency graph of Σ is acyclic, i.e., no predicate of Σ recursively depends on itself. We denote the class of all finite sets of nonrecursive Datalog rules by NRDat.

Given a Datalog rule set Σ, let $Q(x, y)$ be an IDB occurring in Σ. Then, the following rules define the *transitive closure* of Q denoted by Q^+.

$$Q(x, y) \rightarrow Q^+(x, y)$$
$$Q(x, y)^+, Q(y, z) \rightarrow Q^+(x, z)$$

Definition 67. *Generalized regular Datalog* (GRD) is the fragment of Datalog in which recursions are restricted to transitive closure. The class of all finite sets of GRD rules is denoted by GRD.

We are now ready to define *generalized regular queries* (GRQs).

Definition 68. A *generalized regular OMQ* $Q = (\mathbf{S}, \Sigma, q)$ defined over $\mathbf{S}$ is a tuple consisting of a set $\mathbf{S}$ that represents the schema of the input database, a GRD rule set Σ, and a CQ q which is a conjunction of atoms of Σ. We write GRQ for the class of all queries definable by some GRQ query.

Note that in the rest of this chapter, to say that a query Q is a generalized regular query, we may use the notations of $Q \in \mathsf{GRQ}$, and $Q \in (\mathsf{GRD}, \mathsf{CQ})$, interchangeably since they are the same.

Notice that as the following example illustrates, GRD rules may not be connected.

Example 22. *Consider the OMQ* $Q = (\mathbf{S}, \Sigma, q) \in \mathsf{GRQ}$*, where* $\Sigma = \{\sigma_1, \sigma_2\}$*, and* $q = P(x_1, y_1)$*:*

$$\sigma_1 : P(x_1, y_1), Q^+(x_2, y_2), P(x_3, y_3) \rightarrow R(x_1, y_2)$$
$$\sigma_2 : R(x_3, y_3) \rightarrow S(x_3, x_3)$$

It is easy to verify that $Q \in \mathsf{GRQ}$. *Furthermore,* Σ *is not connected.*

In what follows, we introduce a syntactic subset of these rules, which we call k-pseudo-guarded generalized regular Datalog, that is connected by definition, and also offers some computational properties in terms of the complexity of distribution checking. Later in Theorem 82, we exploit these properties to establish the decidability result for checking distribution over components. Before we proceed to define the class of PG2RQ^k queries, let us introduce the novel class of PG rule sets.

Definition 69. (**Pseudo-Guarded**) Given a set A of atoms, an atom $\alpha \in A$ is called *guard* w.r.t. A if $adom(\alpha) = adom(A)$. If the set of guard atoms w.r.t. A is nonempty, then A is called *guarded*. Furthermore, A is called pseudo-guarded if there is a partition $\{A_1, \ldots, A_n\}$ of A such that for each $i \in \{1, \ldots, n\}$, A_i is guarded. We call a set Σ of Datalog rules *pseudo-guarded* (PG) if (i) for each rule $\sigma \in \Sigma$ the set of EDB atoms that occur in the body of σ, when seen as a set of atoms, is pseudo-guarded; and (ii) the body of σ is connected. We denote the class of all finite sets of PG rules by PG.

According to Definition 69, a rule set Σ is PG if the EDB atoms occurring in the body of each rule $\sigma \in \Sigma$ consists of partitions each one of which has a guard atom and furthermore, $body(\sigma)$ is connected. It is clear that PG does not require a single guard for all variables occurring in the body of each rule. However, any guarded rule set is PG.

Definition 70. A finite set Σ of Datalog rules defined over schema $sch(\Sigma)$ is said to belong to pseudo-guarded generalized regular Datalog ($\mathsf{PG2RD}$) if Σ consists of a set Σ_1 of PG rules and a set Σ_2 of rules such that:

(i) The EDB predicates of Σ_1 (and therefore Σ) belong to $sch(\Sigma)$;

(ii) The dependency graph of Σ_1 is acyclic;

(iii) Σ_2 is a set of rules of the following form: $\{R(x, y) \rightarrow S(x, y); S(x, y), R(y, z) \rightarrow S(x, z)\}$ where $R \in sch(\Sigma_1)$ is a predicate of Σ_1 of arity 2.

We define the set of pseudo-guarded generalized regular ontology-mediated queries similar to Definition 68 as follows.

Definition 71. A *pseudo-guarded generalized regular ontology-mediated query* $Q = (\mathbf{S}, \Sigma, q)$ is a tuple consisting of a data schema $\mathbf{S}$, a PG2RD rule set Σ, together with a CQ q defined over $\mathbf{S} \cup sch(\Sigma)$, which is a conjunction of atoms of Σ over $\mathbf{S} \cup sch(\Sigma)$. We write $\mathsf{PG2RQ}$ for the class of all queries definable by some PG2RQ query.

From Definition 69, it is clear that GRD extends NRDat.[1] Furthermore, it is clear that GRQ is at least as expressive as $\mathsf{PG2RQ}$.

Remark 2. *From Definition 70 it can be observed that we can introduce rule sets that belong to* $\mathsf{PG2RD}$ *which are neither guarded nor sticky. As an example, consider the rule set* Σ_2 *of the Introduction Chapter (Chapter 1) which is in* $\mathsf{PG2RD}$ *but is neither guarded nor sticky.*

We are now in a position to formally introduce the intended class of k-pseudo-guarded generalized regular queries.

Definition 72. Let $Q = (\mathbf{S}, \Sigma, q(\mathbf{x})) \in \mathsf{PG2RQ}$ where $q(\mathbf{x}) = \exists \mathbf{y} \phi(\mathbf{x}, \mathbf{y}) \in \mathsf{CQ}$, and $\Sigma' = \Sigma \cup \{\phi(\mathbf{x}, \mathbf{y}) \rightarrow Ans(\mathbf{x})\}$, in which Ans is a fresh predicate of arity $n = |\mathbf{x}|$. Assume that the maximum length of a directed path from some EDB predicate $E \in sch(\Sigma')$ to the Ans predicate in the dependency graph of Σ' is a fixed integer k. Let us call the above length minus 1, the *depth* of Q. Then, we call such a query Q with the depth of k, k-pseudo-guarde generalized regular (PG2RQ^k). We write $\mathsf{PG2RQ}^k$ for the class of all queries definable by some PG2RQ^k query.

7.3 Distribution over Components for GRQ and PG2RQ^k

In this section we dive into the theory of distribution over components for GRQ and $\mathsf{PG2RQ}^k$ fragments. Notice that the characterization provided in

[1] Recall that NRDat is the class of all finite sets of nonrecursive Datalog rules.

this section is of a semantic nature, and does not directly lead to an algorithm to decide whether a query in either GRQ or $\mathsf{PG2RQ}^k$ is distributable or not. However, as we will explain in the next section, using semantic characterizations provided in this section we will be able to construct algorithms to decide whether a given OMQ which belongs to GRQ or $\mathsf{PG2RQ}^k$, distributes over connected components of the input database. To start, we need to proceed with the following proposition for which the proof of Proposition 3 of [32] can be directly adopted. In fact, as shown therein, this proposition can be generalized to all OMQs composed of the class of all finite sets of TGDs and the class of all queries definable by some conjunctive query.

Proposition 73. *For* $\mathsf{Q} \in \{\mathsf{GRQ}, \mathsf{PG2RQ}^k\}$, $\mathsf{conQ} \preceq \mathsf{Q} \cap \mathsf{DIST}$.

Proof. (sketch) Let $Q = (\mathbf{S}, \Sigma, q) \in \mathsf{Q}$. The claim holds because we can always partition the result of the chase of a given $\mathbf{S}$-database D with m partitions $D_1, \ldots, D_m$ and Σ, such that each partition I_i $(1 \leq i \leq m)$ of the result depends only on D_i. □

Proposition 73 implies that connectedness for an OMQ $\mathsf{Q} \in \{\mathsf{GRQ}, \mathsf{PG2RQ}^k\}$ ensures that Q distributes over components.

In order to establish semantic characterizations on distribution over components for GRQ queries, we adopt the transformation used in the connecting operator originally introduced in [8] in the context of Datalog queries with stratified negation (cf., Algorithm 7.1). This operator takes as input an OMQ $Q = (\mathbf{S}, \Sigma, q)$ belonging to a particular class, and outputs another OMQ $Q' = (\mathbf{S}, \Sigma', q')$ which is defined as follows. Σ' is the union of the connected version of Σ and some auxiliary rules to annotate each atom of each component α of the input database with all the constants occurring in it. Moreover, q' is the connected version of q.

For the purpose of constructing auxiliary rules, as illustrated in Algorithm 7.1 (lines 4-11), we introduce a fresh binary predicate con with arity two which returns true if its arguments belong to the same component. It is clear that con is transitive and symmetric. This fact is reflected in the transitivity and symmetry rules which define con. Additionally, $\mathsf{con}(x_1, v), R(x_1, \ldots, x_n) \rightarrow$

$R^*(v, x_1, \ldots, x_n)$ is added to annotate each atom of each component A of the input database with all constants in A.

We construct the connected version of each rule $\sigma \in \Sigma$ in lines 12-19 of Algorithm 7.1, in which each atom $R(x_1, \ldots, x_n) \in \sigma \in \Sigma$ is replaced by a fresh atom $R(v_\sigma, x_1, \ldots, x_n)$ specific to σ.

Finally, we construct the connected version of q in line 21 of Algorithm 7.1.

Overall, Q^{c} is the connected version of Q. This means that whether Q consists of either a single or multiple components, when Q is closed under connecting, Q^{c} always involves a single component. This task is done by introducing fresh predicates which are obtained by adding fresh variables specific to each rule $\sigma \in \Sigma$ or query q to the predicates of σ or q to ensure all rules and the query of the resulting OMQ Q^{c} are connected by means of the introduced fresh variables. Then, for such an input query Q, if the resulting query Q^{c} is equivalent to Q, we can conclude that Q is distributable, as it is equivalent to a connected one.

It was shown in [32] that the membership of an input query Q in DIST implies the equivalence of Q and Q^{c}. For all rule sets $\Sigma \in \mathcal{C}$ and for some class $\mathcal{C}$ of rules that are closed under this operator, i.e., $\mathsf{c}(\Sigma) \in \mathcal{C}$, Proposition 74 holds.

Proposition 74. [32] *For any $\mathcal{C} \subseteq \mathsf{TGD}$ such that $\mathcal{C}$ is closed under connecting operator, $(\mathcal{C}, \mathsf{CQ}) \cap \mathsf{DIST} \preceq (\mathsf{con}\mathcal{C}, \mathsf{conCQ})$.*

Proposition 74 intuitively says that distribution over connected components of the input database for any OMQ which is closed under the connecting operator, and consists of a rule set $\Sigma \in \mathsf{TGD}$ and a conjunctive query q implies connectedness for $(\mathbf{S}, \Sigma, q)$.

By following Algorithm 7.1, it can be seen that GRQ is closed under the connecting operator. Therefore, according to Propositions 73 and 74, we can establish distributability result for GRQ queries as follows.

Proposition 75. $(\mathsf{GRD}, \mathsf{CQ}) \cap \mathsf{DIST} = (\mathsf{conGRD}, \mathsf{conCQ})$.

On the other hand, $\mathsf{PG2RQ}^k$ queries are not closed under connecting. However, we can still establish the same result of Proposition 75 for this class of

Algorithm 7.1 Connecting Operator for GRQ queries

Input: An OMQ $Q = (\mathbf{S}, \Sigma, q) \in \mathsf{GRQ}$;
Output: The transformed OMQ $Q^{\mathsf{c}} = (\mathbf{S}, \mathsf{c}(\Sigma), \mathsf{c}(q))$; A Boolean value $IsClosed$;

1: **procedure** Q^{c}
2: **end procedure**
3: *bool* $IsClosed \leftarrow False$;
4: $\mathsf{c}(\Sigma) = \varnothing$;
5: **for** each predicate $R \in \mathbf{S}$ with arity n **do**
6: **for** each $i, j \in \{1, \ldots, n\}$ **do**
7: $\mathsf{c}(\Sigma) = \mathsf{c}(\Sigma) \cup \{R(x_1, \ldots, x_n) \rightarrow \mathsf{con}(x_i, x_j)\}$, where con is a fresh binary predicate;
8: **end for**
9: $\mathsf{c}(\Sigma) = \mathsf{c}(\Sigma) \cup \{\mathsf{con}(x_1, v), R(x_1, \ldots, x_n) \rightarrow R^*(v, x_1, \ldots, x_n)\}$, where $v \notin \{x_1, \ldots, x_n\}$ and R^* is a fresh predicate of arity $n + 1$;
10: **end for**
11: $\mathsf{c}(\Sigma) = \mathsf{c}(\Sigma) \cup \{\mathsf{con}(x, y) \rightarrow \mathsf{con}(y, x); \mathsf{con}(x, y), \mathsf{con}(y, z) \rightarrow \mathsf{con}(x, z)\}$;
12: $\Sigma' = \varnothing$;
13: **for** each $\sigma \in \Sigma$ **do**
14: **for** each atom $R(x_1, \ldots, x_n) \in \sigma$ **do**
15: Replace $R(x_1, \ldots, x_n)$ with $R^*(v_\sigma, x_1, \ldots, x_n)$, where $v_\sigma \notin \{x_1, \ldots, x_n\}$;
16: **end for**
17: Denote the resulting rule by σ';
18: $\Sigma' = \Sigma' \cup \{\sigma'\}$;
19: **end for**
20: $\mathsf{c}(\Sigma) = \mathsf{c}(\Sigma) \cup \Sigma'$;
21: For each query q, construct $\mathsf{c}(q)$ by replacing each atom $R(x_1, \ldots, x_n)$ with $R^*(v_q, x_1, \ldots, x_n)$;
22: $Q^{\mathsf{c}} \leftarrow (\mathbf{S}, \mathsf{c}(\Sigma), \mathsf{c}(q))$;
23: **if** $\mathsf{c}(\Sigma) \in \mathsf{GRD}$ **then**
24: **return** $\neg IsClosed$;
25: **end if**
26: **return** $IsClosed$;

queries. This exploits the fact that rules in PG2RQ^k queries are connected by definition. Let us prove a lemma.

Lemma 76. $\mathsf{PG2RD} = \mathsf{conPG2RD}$.

Proof. From Definition 70, any given rule set $\Sigma \in \mathsf{PG2RD}$ consists of a set Σ_1 of nonrecursive PG rules and a set Σ_2 of TC rules. Σ_2 is connected by

definition. Therefore it remains to show the connectedness of rules in Σ_1. Based on Definition 69, for a rule set Σ to belong to PG, the bodies of all rules $\sigma \in \Sigma$ must be pseudo-guarded. This means that for each such rule σ there must be a partition $\{A_1, \ldots, A_n\}$ of $body(\sigma)$ such that each partition A_i is guarded, and every pair of partitions share at least one variable. Based on the definition of the partition, it is clear that $\bigcup_{i=1}^{n} A_i = body(\sigma)$, and therefore $\bigcup_{i=1}^{n} var(A_i) = var(body(\sigma))$. Since each partition A_i has (at least) one guard atom, each A_i is connected for $i \in \{1, \ldots, n\}$. Let us denote a guard atom of each partition A_i with A_i^g. Since for each pair of guard atoms A_i^g and A_j^g (of two different partitions), $var(A_i^g) \cap var(A_j^g) \neq \varnothing$, for each rule $\sigma \in \Sigma$ there is a sequence of atoms in $body(\sigma)$ (namely $A^g = \{A_1^g, \ldots, A_n^g\}$) which covers all terms of $body(\sigma)$, and each pair of atoms of A^g share at least one term. Therefore, Σ is connected. □

Remark 3. *From Definition 72, it is clear that each PG2RQ^k query is composed of a PG2RD rule set and a CQ which satisfies the particular depth requirement as defined therein. Alternatively,* $(\mathsf{PG2RD}, \mathsf{CQ})^k$ *is the class of all queries definable by some PG2RQ^k query.*

From Lemma 76, PG2RD rules are connected. Moreover, from Remark 3, connectedness of PG2RD rules can help us establish the same result for the rules of PG2RQ^k queries.

Next, we show another result which along with Lemma 76 can help us establish distributability results for PG2RQ^k queries.

Lemma 77. ([32]) *For all subsets $\mathcal{C}$ of Datalog rule sets that are connected,* $(\mathcal{C}, \mathsf{CQ}) \cap \mathsf{DIST} \preceq (\mathcal{C}, \mathsf{conCQ})$ *(i.e., distribution over components for* $(\mathcal{C}, \mathsf{CQ})$ *implies connectedness of these queries).*

By exploiting Lemmas 76 and 77, we can establish distributability results for PG2RQ^k queries as follows.

Proposition 78. $(\mathsf{PG2RD}, \mathsf{CQ})^k \cap \mathsf{DIST} = (\mathsf{PG2RD}, \mathsf{conCQ})^k$.

7.4 Deciding Distributability of $\mathsf{PG2RQ^k}$ and GRQ via Containment

In previous sections, we provided a semantic characterization for determining the distributable fragments of OMQs consisting of GRQ as well as PG2RQ^k queries. Note that this does not directly lead to a concrete algorithm for deciding distribution over connected components of databases for all fragments of these queries. The reason is that to show the following direction: $(\mathcal{C}, \mathsf{CQ}) \cap \mathsf{DIST} \preceq (\mathsf{con}\mathcal{C}, \mathsf{conCQ})$, i.e., $(\mathsf{con}\mathcal{C}, \mathsf{conCQ})$ for $\mathcal{C} \in \{\mathsf{GRD}, \mathsf{PG2RD}\}$ is at least as expressive as $(\mathcal{C}, \mathsf{CQ}) \cap \mathsf{DIST}$, we need to prove that for each query $q_1 \in (\mathcal{C}, \mathsf{CQ}) \cap \mathsf{DIST}$, there is some query $q_2 \in (\mathsf{con}\mathcal{C}, \mathsf{conCQ})$, such that q_1 and q_2 are equivalent, i.e., for every database D, $q_1(D) = q_2(D)$.

Therefore, if we find a procedure which solves the latter problem in a finite amount of time, then that procedure leads to a concrete algorithm. Note that finding such an algorithm hinges on the decidability of the query equivalence problem which is a known undecidable problem for arbitrary Datalog queries [133].

In this section, we provide mechanisms to decide the membership in DIST for the above query fragments. For this purpose, we distinguish the two cases of GRQ and PG2RQ^k queries, as they have different properties.

Let us start with GRQ queries. In the previous section we showed that these queries are closed under connecting. Therefore, for each OMQ $Q = (\mathbf{S}, \Sigma, q) \in \mathsf{GRQ}$, the transformed OMQ $Q^{\mathsf{c}} = (\mathbf{S}, \mathsf{c}(\Sigma), \mathsf{c}(q))$, as obtained in Algorithm 7.1, belongs to GRQ.

For all distributable OMQs $Q \in \mathsf{GRQ} \cap \mathsf{DIST}$, we have $Q \equiv Q^{\mathsf{c}}$, i.e., distribution over components for GRQ queries implies connectedness of these queries. Furthermore, based on Proposition 73, connectedness of GRQ queries implies their distribution over components. Therefore, if we can show that Q and Q^{c} are equivalent, then, Q is distributable, and any procedure to solve the above equivalence problem leads to a decision problem for membership checking of GRQ in DIST. From [126], it is known that the equivalence problem for GRQ queries is decidable in 2ExpSpace. Henceforth, given that the size of Q^{c}

as computed by Algorithm 7.1 is polynomial in the size of the given OMQ $Q \in \mathsf{GRQ}$, checking whether Q belongs to DIST remains within 2ExpSpace too.

Theorem 79. *Given a query $Q \in \mathsf{GRQ}$, the complexity of checking whether $Q \in \mathsf{DIST}$ is upper bounded by* 2ExpSpace.

For testing query containment of GRQ queries or any of its subsets considered in this chapter we apply the algorithm provided in Section 4.2 of [126] to check the containment of $Q, Q^{\mathsf{c}} \in \mathsf{GRQ}$, and answer membership of Q in DIST positively, if $Q \equiv Q^{\mathsf{c}}$ (i.e., Q is contained in Q^{c}, and Q^{c} in Q).[2]

Let us turn our attention to PG2RQ^k queries. First, note that since these queries are not closed under connecting, therefore, for these queries we cannot utilize the approach of GRQ. From [2], it is known that for each NRDat rule set there is a query answering-preserving transformation to a UCQ, or in other words each rule set in NRDat is UCQ-rewritable. Using this fact, in [32] it was shown that checking distribution over components is decidable (and *co*NExpTime-complete) for NRDat, even for fixed schema arity.

It is not hard to establish that the data complexity of reasoning with $\mathsf{PG2RQ}^k$ queries is NLogSpace-complete. The upper bound comes from that of GRQ queries which is known to be in NLogSpace [126]. Moreover, to prove the hardness, we utilize a reduction from the evaluation problem for *linear Datalog* which is a NLogSpace-complete problem [55]. Since (i) the evaluation problem of all UCQ-rewritable queries is in AC^0 in data complexity; and (ii) AC^0 is strictly contained in NLogSpace, we conclude that PG2RQ^k queries cannot be UCQ-rewritable. For more information, cf., [11].

Therefore, we cannot conduct our distribution analysis for these rules via the rewriting approach as established in the previous chapter.

For any class of queries Q, let $\mathsf{Dist}(\mathsf{Q})$ denote the problem of checking membership of $Q \in \mathsf{Q}$ in DIST. In this section, we show that $\mathsf{Dist}(\mathsf{PG2RQ}^k)$

[2]Note that in [126] it is assumed that each GRQ query is encoded in an equivalent query language known as the *nested union of conjunctive two-way regular path query* (nUC2RPQ) since there are well established automata-theoretic techniques for the latter class of queries that can be applied to obtain containment results for GRQs.

is decidable, and then, we elaborate on how to decide whether $\mathsf{Dist}(\mathsf{PG2RQ}^k)$ returns a true/false answer. To tackle the distribution problem for OMQs consisting of these fragments, we exploit the fact that as argued in Lemma 76, the rules of $\mathsf{PG2RQ}^k$ queries are by definition connected.

Recall that Proposition 78 shows that distribution over components implies connectedness for PG2RQ^k queries.

Consider an OMQ $Q = (\mathbf{S}, \Sigma, q) \in \mathsf{PG2RQ}^k \cap \mathsf{DIST}$, where q in defined by the following CQ: $\exists \mathbf{y} \phi(\mathbf{x}, \mathbf{y})$. Obviously, Q is either unsatisfiable or satisfiable. The former means that for all databases D defined over $\mathbf{S}$, $Q(D) = \varnothing$. For this case, this proposition holds trivially as we can choose an arbitrary unsatisfiable query from $\mathsf{conPG2RQ}^k$ to verify the claim.

For the latter case (i.e., when Q is satisfiable), let us denote the set of components of q with $\{\phi_1, \ldots, \phi_m\}$. If $m = 1$, the claim holds trivially as in this case, connectedness holds for $\mathsf{PG2RQ}^k$ unconditionally, and so does the claim. So, to proceed, let us focus on $m \geq 2$. We claim that one can focus on only one single component of q in order to construct an OMQ which is equivalent to Q. Let us denote this component using q_i for some $i \in \{1, \ldots, m\}$, for which we can construct $Q_i = (\mathbf{S}, \Sigma, q_i)$. Then since Q_i is connected by definition, this particular Q_i satisfies Proposition 78.

To proceed further, let us consider two cases for the CQ: q is either Boolean or nonBoolean. In the former, for the arity n of q we have: $n = 0$, and in the latter, $n > 0$.

In the latter case since Σ is connected, it can be shown that all answer variables of q must occur in a single component as otherwise, Q will be unsatisfiable which contradicts our initial assumption. This leads to the conclusion that we can focus on the single component $q_{\mathbf{x}}$ of q which possesses all answer variables of q. Let $Q_{\mathbf{x}} = (\mathbf{S}, \Sigma, q_{\mathbf{x}})$. Based on what was argued above we conclude that $Q_{\mathbf{x}} \in \mathsf{conPG2RQ}^k$, and we have: $Q \equiv Q_{\mathbf{x}}$ as desired.

In the former case where q is Boolean, we proceed by following the same goal: To show that there is a component q_i of q, in which $i \in \{1, \ldots, m\}$ such that $Q_i = (\mathbf{S}, \Sigma, q_i) \equiv Q$. This can be done by finding an integer $i \in \{1, \ldots, m\}$ such that $Q_i \subseteq Q_i^-$, where Q_i^- is the query obtained from Q by removing the

component ϕ_i of q. Then (i) $Q_i \subseteq Q_i^-$; (ii) $Q_i^- \subseteq Q$; and (iii) $Q \subseteq Q_i$, lead to the conclusion that Q and Q_i are equivalent, or in other words, there is an integer $i \in \{1, \ldots, m\}$ such that $Q \equiv Q_i$. Finally, since this Q_i is connected by definition, it satisfies Proposition 78 as desired.

Based on the above arguments, it is easy to show the following Lemma.

Lemma 80. *Given an OMQ $Q = (\mathbf{S}, \Sigma, q) \in \mathsf{PG2RQ}^k$, let $\{q_1, \ldots, q_m\}$ be the set of components of q. We have $Q \in \mathsf{DIST}$ if and only if one of the following conditions holds:*

(i) *Q is not satisfiable;*

(ii) *There is an integer $i \in \{1, \ldots, m\}$ such that $Q_i = (\mathbf{S}, \Sigma, q_i) \in \mathsf{PG2RQ}^k$, and $Q_i \subseteq Q$.*

Notice that checking condition (i) of Lemma 80 boils down to checking the containment problem for $\mathsf{PG2RQ}^k$ queries. Therefore, if we solve the query containment problem for any query $Q \in \mathsf{PG2RQ}^k$, then, we have a decision algorithm for checking distributability of Q. In the sequel, we present a positive answer to the containment problem of these queries.

Given a query language $\mathcal{Q}$, the containment problem for $\mathcal{Q}$, denoted $\mathsf{Cont}(\mathcal{Q})$, takes as input two queries $Q_1, Q_2 \in \mathcal{Q}$, and checks whether for all databases D, $Q_1(D) \subseteq Q_2(D)$, where queries in Q_1 and Q_2 are of the same arity.[3]

Proposition 81. $\mathsf{Cont}(\mathsf{PG2RQ}^k)$ *is decidable.*

Proof. The above result is shown by a reduction to the containment of the bounded depth GRQ queries as studied in [126]. It is known that the query containment problem for these queries is ExpSpace-complete. We utilize the same construction provided in [139] for rewriting a query $Q \in \mathsf{GRQ}$ into an equivalent Datalog query $Q_{Dat} = (\Pi, Ans)$, where Π is a set of standard nonrecursive Datalog rules of arbitrary arity with some additional rules that define transitive closure of binary relations.

[3]Recall that each OMQ $Q = (\mathbf{S}, \Sigma, q)$ is interpreted as a query over $\mathbf{S}$ the arity of which is that of q.

Notice that as discussed in [139], this restriction of recursion in Datalog is the basis for the decidability of query containment for GRQ. Furthermore, GRQ generalizes the class of *regular queries* RQ in which the queries are defined on graph databases, and therefore all rules of Π are binary. However, this restriction does not change the complexity of containment checking for these queries, since relations of arbitrary arity can be easily encoded to binary relations [126]. This immediately leads to the conclusion that bounding the maximum arity of rules does not change the complexity of the corresponding containment problem. □

Exploiting Proposition 81, we are now ready to present the main result of this section.

Theorem 82. *The complexity of checking* $\mathsf{Dist}(\mathsf{PG2RQ}^k)$ *is in* ExpSpace.

7.5 The Landscape of Distributable OMQs

The fragments we investigate in this chapter for checking distributability over connected components so far belong to the following two general categories: (1) Those for which the corresponding fragment is UCQ-rewritable; and (2) Those with decidable problem of equivalence (i.e., decidable OMQ containment problem).

In the sequel, we show that the decidability of the problem of checking distribution over components can be ubiquitously applied to OMQs constructed from other query languages which belong to the first or the second category above. In case the query is not UCQ-rewritable, but the problem of equivalence of the query language is decidable, we show that there is a fragment in that language which has a decidable problem of checking distribution over components. Before we proceed, let us introduce an auxiliary notion.

7.5.1 Datalog with Negation

A Datalog rule with negation is a rule σ of the form $\phi_{pos}(\mathbf{x},\mathbf{y}) \wedge \phi_{neg}(\mathbf{x},\mathbf{y}) \rightarrow \psi(\mathbf{x})$ in which $\phi_{pos}(\mathbf{x},\mathbf{y})$ and $\phi_{neg}(\mathbf{x},\mathbf{y})$ are conjunctions of atoms, and $\psi(\mathbf{x})$ is

an atom. We call $\psi(\mathbf{x})$ *the head of* σ, and $\phi_{pos}(\mathbf{x},\mathbf{y})$ and $\phi_{neg}(\mathbf{x},\mathbf{y})$ are called *positive body atoms* and *negative body atoms*, respectively. We only consider rules σ in which each variable of $\psi(\mathbf{x})$ and $\phi_{neg}(\mathbf{x},\mathbf{y})$ also occurs in $\phi_{pos}(\mathbf{x},\mathbf{y})$. Let $\phi_{pos} = \{A_1,\ldots,A_n\}$ and $\phi_{neg} = \{B_1,\ldots,B_m\}$ and $\psi = H$. Then a rule σ as defined above may be written in a conventional syntax as follows:

$$A_1(\mathbf{x},\mathbf{y}),\ldots,A_n(\mathbf{x},\mathbf{y}),\neg B_1(\mathbf{x},\mathbf{y}),\ldots,\neg B_m(\mathbf{x},\mathbf{y}) \rightarrow H(\mathbf{x})$$

We say that a Datalog rule with negation σ is over a schema if all of its atoms be over that schema. A *Datalog with negation* Σ over a schema $sch(\Sigma)$ is a set of Datalog rules with negation over $sch(\Sigma)$.

The notions of intensional/extensional predicates and atoms of a Datalog with negation are defined analogous to those of a Datalog. A stratification of a Datalog with negation Σ is a partition of Σ into a set $\{\Sigma_1,\ldots,\Sigma_m\}$ of Datalog rules such that the following conditions are satisfied:

1. All rules $\sigma \in \Sigma$ that have the occurrence of the same IDB predicate in $head(\sigma)$ are in the same partition;

2. If an IDB predicate R_1 occurs positively in the body of a rule with head R_2, R_1 should occur in the head of a rule which appears in a partition with index smaller than or equal to where R_2 is defined;

3. If an IDB predicate R_1 occurs negative in the body of a rule with head R_2, R_1 should occur in the head of a rule which appears in a partition with index smaller than where R_2 is defined;

A Datalog with negation is called *stratified* if there is a stratification for it. Sometimes we call such a set *Datalog with stratified negation*. We write $\mathsf{Datalog}^{\mathrm{StrNeg}}$ for the class of all finite sets of Datalog rules with stratified negation.

Let Σ be a $\mathrm{Datalog}^{\mathrm{StrNeg}}$ rule set, and Σ' be Σ with all the negative body atoms removed. Σ is called *connected* if Σ' is connected.

Let $\mathcal{C}$ be a subclass of $\mathrm{Datalog}^{\mathrm{StrNeg}}$ rules. We denote by $\mathsf{con}\mathcal{C}$ the class of all finite sets of rules that belong to $\mathcal{C}$ and that are connected.

In the following two subsections we introduce these logics along with results on their combination with transitive closure.

7.5.2 Distribution of Logics without Transitivity

In this section we lay down the landscape of distributability of different logics that are well known in the literature over connected database components, and we derive new results regarding OMQ distribution for these fragments. In order to achieve decidability for all of these fragments, we exploit the tools that were developed in previous sections. Warning and apology: in the rest of this chapter we may use the terms OMQs, logics, fragments, and (query) languages interchangeably.

The logics we consider in this chapter are as follows: Unary-Negation Fragment of First-Order logic (UNFO) [136], *Guarded fragment* (GFO) [9], *Monadic Datalog* (MDL) [77], *Least Fixpoint Logic* (LFP) [142], *Monadically Defined Queries* (MODEQ) [37], *Guarded Negation Datalog* (GNDatalog) [21], *Guarded Negation Fixpoint Logic* (GNFP) [22], *Guarded Negation Fixpoint Logic with unrestricted parameters* (GNFP-UP) [28], C2RPQ [46], UC2RPQ, GRQ [126], RQ [126], *frontier-1 TGD* (fr-1), FGTGD [18], and *First-order Logic* (FO). We adopt the convention similar to the previous classes of OMQs we considered in this chapter to denote the sets of each one of these languages, e.g., GFO is used to denote the sets of GFO OMQs.

For any of these logics, we distinguish the notions of *ontologies* and *queries* constructed from them as follows: For each class X above, an X *ontology* is specified by a finite set of formulas of X. Moreover, an X *query* is a pair (Σ, Ans), where Σ is an X ontology and Ans is a union of conjunctive queries over the schema of Σ.

The containment problem is known to be decidable for the queries constructed from all of these logics with the complexities as listed in Table 7.1 in which decidability results for all queries constructed from ontologies with $\star$ are established only in the presence of answer-guarded conjunctive queries (AGCQs). The same assumption is made to establish all results concerning these fragments in the rest of this chapter. Furthermore, the expressiveness

relations that exist between these logics among others are shown in Figure 7.5.1.

Table 7.1: Containment problem for the logics considered in this chapter

Language	**Complexity of the Containment Problem**
GNDatalog*	2ExpTime [21]
LFP	Undecidable
GFO*	2ExpTime [84]
GNFO*	2ExpTime [20]
UNFO*	2ExpTime [136]
MDL	2ExpTime [29, 52]
MODEQ	2ExpTime [37]
GNFP*	2ExpTime [20]
GNFP-UP*	Nonelementary [28]
fr-1*	2ExpTime [136]
FGTGD*	2ExpTime [136]
C2RPQ	ExpSpace [46]
UC2RPQ	ExpSpace
GRQ	2ExpSpace [126]
RQ	2ExpSpace [126]
FO	Undecidable
Datalog	Undecidable [133]

Definition 83. Each $Q' = (\mathbf{S}', \Sigma', q') \in \mathsf{Q}'$ is called a *standard* Q' *translation* of an OMQ $Q = (\mathbf{S}, \Sigma, q) \in \mathsf{Q}$ if (1) $\mathsf{Q} \preceq \mathsf{Q}'$, and (2) Q' preserves certain answers of Q, i.e., $Q \equiv Q'$. The notion of standard Q' translation can be extended to sets of OMQs in a natural way.

There are known standard $\mathsf{Datalog}^{\mathrm{StrNeg}}$ translations of $\mathsf{Q} \in \{\mathsf{GFO}, \mathsf{fr\text{-}1}, \mathsf{MDL}, \mathsf{FGTGD}, \mathsf{UNFO}, \mathsf{GNFO}, \mathsf{GNDatalog}, \mathsf{MODEQ}, \mathsf{C2RPQ}, \mathsf{UC2RPQ}, \mathsf{RQ}, \mathsf{GRQ}\}$ queries. For a recent survey, refer to [27]. Without loss of generality, in the rest of this chapter, we assume that each query $Q = (\mathbf{S}, \Sigma, q) \in \mathsf{Q}$ where Q is as listed above, is already in the standard translation to $\mathsf{Datalog}^{\mathrm{StrNeg}}$ queries.

Note that with an exception of the following classes: $\{\mathsf{GRQ}, \mathsf{RQ}\}$, the other classes are not closed under connecting operator. Therefore, for these classes we can establish distribution results similar to that of GRQ queries of the previous section.

On the other hand, the following classes of queries: $\{\mathsf{GFO}, \mathsf{MODEQ}, \mathsf{fr}\text{-}1, \mathsf{MDL}$ $\mathsf{C2RPQ}, \mathsf{UC2RPQ}, \mathsf{FGTGD}, \mathsf{UNFO}, \mathsf{GNFO}, \mathsf{GNDatalog}\}$ are not closed under connecting, and therefore, Proposition 74 is not applicable for them. However, if we focus on the connected fragment of the above ontologies, we can establish such result. Then, we can prove the following theorem with a proof similar to that of Lemma 80 and Proposition 75.

Theorem 84. *Let* $\mathsf{C}_1 \in \{\mathsf{GFO}, \mathsf{FGTGD}, \mathsf{UNFO}, \mathsf{GNFO}, \mathsf{GNDatalog}, \mathsf{MDL}, \mathsf{C2RPQ},$ $\mathsf{fr}\text{-}1, \mathsf{MODEQ}, \mathsf{UC2RPQ}\}$ *be a class of ontologies. Then we have:* (i) $(\mathsf{conC}_1, \mathsf{CQ}) \cap$ $\mathsf{DIST} = (\mathsf{conC}_1, \mathsf{conCQ})$*; and* (ii) *if* $\mathsf{C}_2 \in \{\mathsf{GRQ}, \mathsf{RQ}\}$*, then* $(\mathsf{C}_2, \mathsf{CQ}) \cap \mathsf{DIST} =$ $(\mathsf{conC}_2, \mathsf{conCQ})$.

Theorem 84 states that for all queries listed therein, their connected fragment effectively provides a fragment for which the problem of distribution over components is decidable. Before establishing the main result of this section, let us present a lemma.

Lemma 85. *For each query language* $\mathsf{Q} \in \{\mathsf{GFO}, \mathsf{fr}\text{-}1, \mathsf{FGTGD}, \mathsf{UNFO}, \mathsf{GNFO},$ $\mathsf{GNDatalog}, \mathsf{MDL}, \mathsf{MODEQ}, \mathsf{C2RPQ}, \mathsf{UC2RPQ}, \mathsf{RQ}, \mathsf{GRQ}\}$*,* $\mathsf{Cont}(\mathsf{Q})$ *is decidable.*

It is not hard to see that Lemma 80 can be generalized to any query $Q = (\mathbf{S}, \Sigma, q) \in (\mathsf{conDatalog}^{\mathrm{StrNeg}}, \mathsf{CQ})$. The proof is similar to that of Lemma 80. Note that this generalization does not necessarily lead to any decidable procedure for these queries. However, when combined with Lemma 85, the decidability results can be established for the queries mentioned in that lemma.

Theorem 86. *For each class of ontologies* C_1 *and* C_2 *as listed in Theorem 84, the membership checking problem of* (i) $(\mathsf{conC}_1, \mathsf{CQ})$*; and* (ii) $(\mathsf{C}_2, \mathsf{CQ})$ *queries in* DIST *is decidable, and is in elementary for all of the above queries. The exact complexity upper bound for each language is summarized in Figure 7.5.1.*

In Figure 7.5.1, the considered classes of OMQs with the complexity of deciding distribution over components for each fragment is demonstrated. The lines in the figure represent containment at the level of expressiveness.

For each query $\mathsf{Q}_1 \in \{\mathsf{GFO}, \mathsf{fr}\text{-}1, \mathsf{FGTGD}, \mathsf{MODEQ}, \mathsf{C2RPQ}, \mathsf{UC2RPQ}, \mathsf{UNFO},$ $\mathsf{MDL}, \mathsf{GNFO}, \mathsf{GNDatalog}\}$, based on Theorem 86, there is a fragment for which

the problem of checking distribution over components has the same complexity upper bound as that of the containment checking problem for the corresponding fragment Q_1. Moreover, for $\mathsf{Q}_2 \in \{\mathsf{RQ}, \mathsf{GRQ}\}$, the same argument holds for the entire fragment.

For any query language Q with ** as a superscript (i.e., Q^{**}) in Figure 7.5.1, its connected fragment has a decidable problem of distribution over components according to the proof technique of Theorem 86. In addition, for those logics without **, the above problem has a complexity upper bound shown below its name for all OMQs constructed in that language. The techniques developed in this chapter cannot provide any result for the problem of distribution over components for fragments with "?" under their name.

Bidirectional lines in Figure 7.5.1 are to denote that the languages they connect are equi-expressive. In particular, $\mathsf{NRDat} = \mathsf{UCQ}$ [2], and TGD with FC $=$ $\mathsf{Datalog}$, where FC stands for the *finite chase*, and TGD with FC is the class of all finite sets of TGDs that have finite chase (aka *finite expansion sets*, or FES), cf., [18]. The latter is a recent result shown in [104].

7.5.3 Distribution of Logics Combined with Transitivity

A serious weakness of most logics that we studied in the previous section, with regard to their expressive power, is their inability to express transitivity of a binary relation. In previous sections, we discussed the consequences of this limitation in the knowledge representation or databases applications in which relations such as *part-of* or *stronger-than* occurs frequently.

As a notational convention, for each class of OMQs $\mathcal{Q}$ with $\mathcal{Q}+\mathsf{TC}$, we denote combination of the transitivity operator with $\mathcal{Q}$. It is known from the literature that adding transitivity to most logics leads to undecidability of the satisfiability and containment problem of the underlying logics.

We show how this will affect the problem of distribution over components, and then we single out classes of OMQs for which this problem is decidable. Before we can present the results of this section, let us introduce some notions.

A *transitivity rule* is a Datalog rule of the form $P(x, y), P(y, z) \rightarrow P(x, z)$. A predicate is called *transitive* if it appears in a transitivity rule.

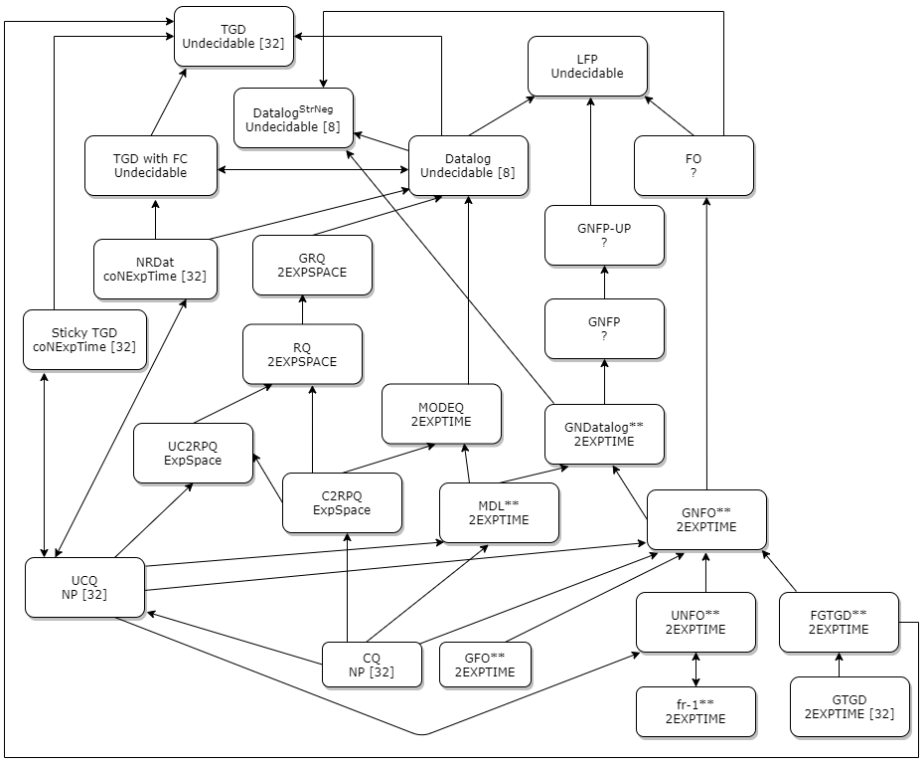

Figure 7.5.1: Distribution checking for different logics.

Let $Q = (\mathbf{S}, \Sigma, q) \in \mathcal{Q}$ be such that $\mathcal{Q} \preceq \mathsf{Datalog}^{\mathrm{StrNeg}}$. Let $\Sigma{+}\mathsf{TC}|_P$ denote the rule set which is the result of combining Σ with transitivity rules for some binary predicates in $P \subseteq sch(\Sigma)$. We define:

$$\mathcal{Q}{+}\mathsf{TC} = \Big\{ \bigcup_{\forall \Sigma \in \mathcal{Q}} \big[\Sigma{+}\mathsf{TC}|_P\big] \text{ for some binary predicates } P \subseteq sch(\Sigma) \Big\}$$

i.e., $\mathcal{Q}{+}\mathsf{TC}$ denotes the class of OMQs obtained by adding zero or more transitivity rules to rule sets Σ in $\mathcal{Q}$.

For each class of queries Q we studied in the previous section, it is clear that

$$\mathsf{Q} \preceq \mathsf{Q}{+}\mathsf{TC} \preceq \mathsf{Datalog}^{\mathrm{StrNeg}}{+}\mathsf{TC} = \mathsf{Datalog}^{\mathrm{StrNeg}} \tag{7.5.1}$$

Note that adding TC to RQ and GRQ does not increase expressivity of the language, as the original fragments already allow for transitivity, i.e., $\mathsf{GRQ} = \mathsf{GRQ}{+}\mathsf{TC}$.

Based on (7.5.1), exploiting the fact that transitivity rules are connected by definition, it is clear that combining any query language Q with TC does not

change the syntactic connectivity status of Q. On the other hand, since adding TC to any query language may result in undecidability of the satisfiability and equivalence problem of the resulting language, it turns out that for all of the studied query languages where the combination is undecidable, the problem of distribution over connected components will be undecidable too. This holds for the following fragments and all other ones that are strictly more expressive than these classes:

$$\{\mathsf{GFO{+}TC}, \mathsf{GTGD{+}TC}, \mathsf{GNFO{+}TC}, \mathrm{Sticky}\mathsf{TGD{+}TC}\}$$

On the other hand, the problem of containment is known to be decidable for the following fragments which permit transitivity, as well as any other class of languages that is not strictly more expressive than the following:

$$\{\mathsf{UNFO{+}TC}\ [58], \mathsf{RQ}, \mathsf{GRQ}\ [126]\}$$

Since adding transitivity does not change the syntactic connectivity of the queries, we can exploit the same tools that were used in the previous section for the new fragments. Therefore, we have similar results with the previous section which is stated in the following Theorem.

Theorem 87. *For* $\mathsf{Q} \in \{\mathsf{UNFO{+}TC}, \mathsf{RQ}, \mathsf{GRQ}\}$*, case* (ii) *of Theorems 84 and 86 holds. The exact complexity upper bounds for the resulting languages are summarized in Figure 7.5.2.*

In Figure 7.5.2, different classes of query languages along with their complexity of membership checking in DIST is demonstrated. When compared to Figure 7.5.1, it can be seen that for most of OMQs in the figure without transitivity, adding TC leads to undecidability of the problem of checking distribution over components. This holds in particular for the guarded fragment. However, for the unary negation fragment the above statement does not hold. In addition, based on Theorem 87, the problem of checking distributability over components has the same complexity upper bound of the same problem for the fragment without transitivity.

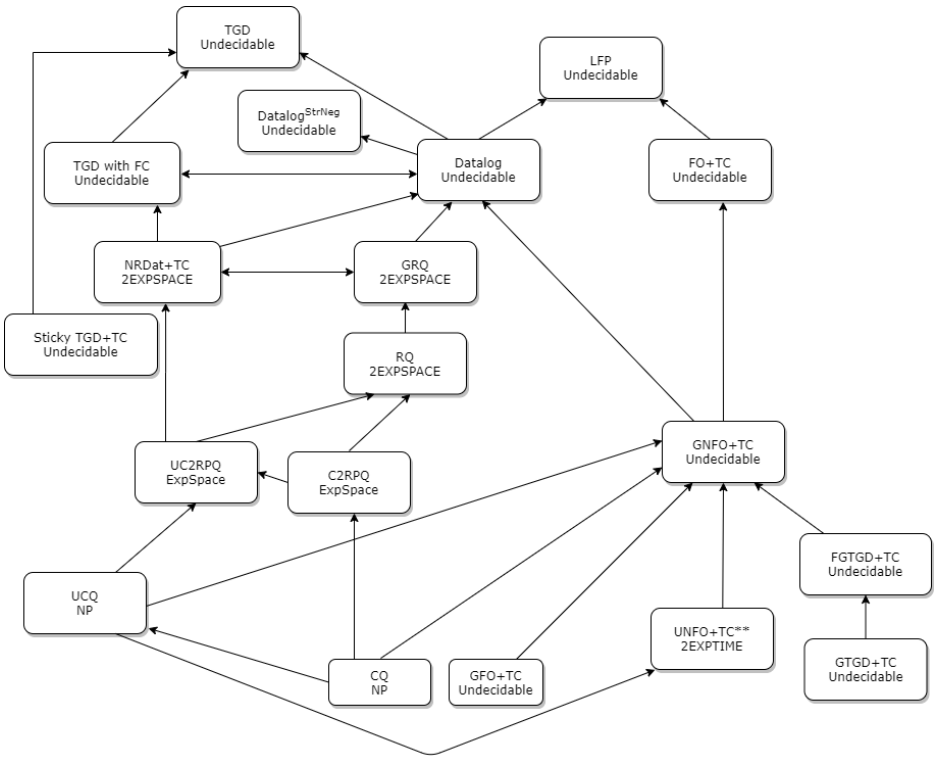

Figure 7.5.2: Distribution checking for different logics combined with transitivity.

7.6 A Case Study

Consider the following rule set $\Sigma = \{\sigma_1, \ldots, \sigma_{k+4}\}$ for some integer $k \geq 1$ which models a secure *multi-party computation* (MPC) network that involves a group of n parties who want to compute a function together.

The MPC scenario we are modelling is based on Shamir's secret sharing scheme (sss) [132], which is common in the MPC literature (cf. [76], and [57] for an implementation of MPC based on sss). In this threshold-based model, a secret is divided into parts giving each party its own unique part (share). A minimum number of shares, called the threshold value, is then needed to reconstruct the original secret.

We make the following assumptions in our model:

1. If some party (also called node in the MPC network) x_i is going to communicate a secret message to another party x_j for some $i, j \in \{1, \ldots, n\}$

where $i \neq j$, a secure channel must be established from x_i to x_j (Schannel(x_i, x_j));

2. There exists an adversary who is corrupted. A corrupted party x_i may corrupt some other party x_j in the network (Corrupts(x_i, x_j));

3. A threshold is an integer denoted by t_i for $i \in \{1, \ldots, k\}$ where $1 \leq t_i \leq n$, and k represents the number of different secrets. For different secrets different thresholds apply. This number represents the number of shares of some secret among parties. We assume that an adversary acts rationally by trying to corrupt as many parties as possible, but only those with whom she shares the ith secret, until she ensures that at least t_i parties with this constraint are corrupted (the fact that Eve shares the ith secret with Alice is denoted by SharesSecr$_i$(Eve, Alice)).

x corrupts y if x is corrupted and furthermore, there is a secure channel via which it can connect through a chain of intermediate nodes that can pass the messages using trusted third-party message-passing nodes (σ_2). In order to establish a successful trusted third-party message passing protocol from a node x to another node y, there must be a secure channel from x to y and a trusted third-party z which acts as an intermediary node to receive messages from x and then deliver them to y (σ_3). Further, such a trust can only be established from x to y if both of these nodes are corrupted and they are connected with the same person via a chain of good friends (σ_4).

If any of Comprom$_i(x_1, \ldots, x_n, t_i)$ for $i \in \{1, \ldots, k\}$ is true which means that Compromised$(x_1, \ldots, x_n)$ returns true, at least t_i parties are corrupted where t_i is the threshold for the ith secret (σ_1). This implies that our MPC network with k secrets and n parties is compromised. Note that σ_1 is equivalent to k rules of the form Comprom$_i(x_1, \ldots, x_n, t_i) \rightarrow$ Compromised$(x_1, \ldots, x_n)$ $(1 \leq i \leq n)$. The disjunctions used there are only for the convenience.

Finally, if there is a chain of t_i parties that corrupt each other and share the ith secret among one another, then the ith secret is compromised (σ_5-σ_{k+4}).

$$\sigma_1 : \big(\text{Comprom}_1(x_1, \ldots, x_n, t_1) \vee \cdots \vee \text{Comprom}_k(x_1, \ldots, x_n, t_k)\big) \rightarrow \text{Compromised}(x_1, \ldots, x_n)$$
$$\sigma_2 : \text{Corrupted}(x), \text{Schannel}(x, z_1), \text{TMPassing}^+(z_1, z_2), \text{Schannel}(z_2, y) \rightarrow \text{Corrupts}(x, y)$$
$$\sigma_3 : \text{Schannel}(x, y), \text{Trusts}(x, z), \text{MPassing}^+(x, z), \text{MPassing}^+(z, y) \rightarrow \text{TMPassing}(x, y)$$
$$\sigma_4 : \text{Corrupted}(x), \text{Corrupted}(y), \text{GoodFriends}^+(x, z), \text{GoodFriends}^+(y, z) \rightarrow \text{Trusts}(x, y)$$

$$\begin{cases} \sigma_5 : \text{SharesSecr}_1^+(x_1, x_2), \\ \quad \text{Corrupts}^+(x_1, x_2), \text{SharesSecr}_1^+(x_2, x_3), \text{Corrupts}^+(x_2, x_3), \ldots, \\ \quad \text{SharesSecr}_1^+(x_{t_1-1}, x_{t_1}), \text{Corrupts}^+(x_{t_1-1}, x_{t_1}) \rightarrow \text{Comprom}_1(x_1, \ldots, x_n, t_1) \\ \quad \vdots \\ \sigma_{k+4} : \text{SharesSecr}_k^+(x_1, x_2), \\ \quad \text{Corrupts}^+(x_1, x_2), \text{SharesSecr}_k^+(x_2, x_3), \text{Corrupts}^+(x_2, x_3), \ldots, \\ \quad \text{SharesSecr}_k^+(x_{t_k-1}, x_{t_k}), \text{Corrupts}^+(x_{t_k-1}, x_{t_k}) \rightarrow \text{Comprom}_k(x_1, \ldots, x_n, t_k) \end{cases}$$

Notice that for each binary predicate P, P^+ is the transitive closure of P, and the TC rules are not explicitly written for this example. It is not difficult to show that $\Sigma \in \mathsf{GRD}$. Furthermore, Σ does not belong to RD due to the presence of predicates of arity greater than 2.

Moreover, let $Q = (\mathbf{S}, \Sigma, q)$ be a query of depth c, where c is a constant and q is a CQ. Then, Q is equivalent to another query $Q' = (\mathbf{S}', \Sigma', q)$ that belongs to $\mathsf{PG2RQ}^c$. Q' is constructed as follows. $\Sigma' = \Sigma \cup \{\text{Corrupted}_E(x) \rightarrow \text{Corrupted}(x); \text{Schannel}_E(x, y) \rightarrow \text{Schannel}(x, y)\}$ and $\mathbf{S}' = \mathbf{S} \cup \{\text{Corrupted}_E/1, \text{Schannel}_E/2\}$, where we write R/n to denote that R has arity n.

It is not hard to show that given a (syntactically) connected query $Q \in \mathsf{GRQ}$ of depth k,[4] where k is an integer, we can always find a query Q' such that (i) $Q \equiv Q'$, and (ii) $Q' \in \mathsf{PG2RQ}^{k+1}$. The construction of such a query is based on introducing a fresh EDB predicate P_E for each EDB predicate P of arity n that occurs in the rule set of Q and add $\{P_E(\mathbf{x}) \rightarrow P(\mathbf{x})\}$ to the rule set of Q to obtain that of Q', where $|\mathbf{x}| = n$.

[4]The depth of GRQ queries is defined analogous to that of PG2RQ queries.

7.7 Experiments on PG2RQ$^{\mathbf{k}}$ queries

The goal of this section is multi-fold: First, we construct OMQs that belong to $\mathsf{PG2RQ}^k$ queries. In the next step, we analyze these queries for membership in DIST, i.e., whether they are distributable over components. Finally, we evaluate the performance of the task of query answering for those queries which are distributable: we establish several optimizations for these queries to improve the above task.

We analyzed 120 (non-Horn) ontologies from the following corpora: BioPortal, the Protege library, Gardiner ontology corpus [71], as well as a number of ontologies from Open Biomedical Ontology (OBO) corpus. In order to transform ontologies into Datalog rules we used KAON2 tool [118]. As a first preprocessing step, we discarded the translated ontologies with those constructs outside of $\mathcal{SHIQ}$, which are therefore outside of what KAON2 can support. As a second preprocessing step we discarded all ontologies with constructs that correspond to disjunctive as well as recursive Datalog.

We ended up with 105 ontologies for which KAON2 succeeded in producing Datalog.

We fixed k to be 10, and then we doctored the resulting ontologies for those without transitive predicates, by adding transitive axioms on top of the those ontologies, randomly, for 3 different IDB predicates in all of the considered corpora.

Among the resulting ontologies we handcrafted conjunctive queries and input databases while ensuring that the depth of these OMQs is bounded by 10. Then, in order to check the membership of each OMQ Q in PG2RQ^{10}, we removed the restriction of the rules that occur in Q to satisfy Condition (i) of Definition 69.

After removing this condition, we end up with connected GRQ queries of bounded depth. Note that this can be done without loss of generality, since, as illustrated in the example of Section 7.6, for each connected OMQ $Q \in \mathsf{GRQ}$ with depth k, one can always find an equivalent OMQ which belongs to PG2RQ^{k+1} for a fixed value of k. Then, we proceeded with checking

Table 7.2: Statistical results for distributability membership checking

# Total queries	# PG2RQ^{10} queries	# Dist. queries	Avg. dist. checking time (s)
105	96	62	242.7

membership of the resulting OMQs in DIST according to Algorithm 7.2.

After accomplishing the above tasks, we found 96 (91.4%) OMQs that belong to PG2RQ^{10}.

Algorithm 7.2, which is based on Lemma 80, takes as input, an OMQ Q that belongs to PG2RQ^k, and outputs a Boolean value that evaluates to true when Q is distributable and false, otherwise.

Algorithm 7.2 Checking membership of $Q = (\mathbf{S}, \Sigma, q) \in$ PG2RQ^k in DIST

Input: An OMQ $Q \in \mathsf{Q} =$ PG2RQ^k;
Output: Boolean value IsDistributable;

```
1: procedure Dist(Q), WHERE Q ∈ Q
2: end procedure
3: IsDistributable = false;
4: {q_1, ..., q_m} ← Components of q;
5: if Q is unsatisfiable then
6:     IsDistributable = true;
7:     return IsDistributable;
8: end if
9: for Each integer i ∈ {1, ..., m} do
10:    if Q_i = (S, Σ, q_i) ∈ PG2RQ^k and Q_i ⊆ Q then
11:        IsDistributable = true;
12:        return IsDistributable;
13:    end if
14: end for
15: return IsDistributable;
```

In the corpora under consideration we discovered 62 ontologies (64.6%) that were distributable. Statistics related to this analysis is summarized in Table 7.2.

Additionally, for distributable queries we evaluated the performance of query answering using two Datalog engines of RDFox and Graal. The results of this evaluation is summarized in Table 7.3.

Table 7.3: Statistics of distributed versus centralized query answering

# Experimented queries	Centr.		Distr.		Avg. # of components
	RDFox	Graal	RDFox	Graal	
62	73.2	248.1	27.3	86.6	3

Query answering optimization

In a nutshell, a query $Q \in \mathsf{PG2RQ}^k$ that is distributable lends itself to query optimizations that are available for free. For a simple explanation, notice that for such queries, according to Lemma 80, one is able to exploit the (query) results of only one component Q_i of Q to answer Q. This can be a huge advantage for answering queries with many components.

In this section by focusing on dyadic OMQs,[5] we establish a number of further optimizations which can be applied during the course of query answering for queries which conform to particular characteristics which we describe in the next lemma. This way, one is able to improve the performance of query answering even further.

Lemma 88. *Let $Q^1 = (\mathbf{S}, \Sigma, q^1)$, $Q^2 = (\mathbf{S}, \Sigma, q^2)$, and D be a $\mathbf{S}$-database. Let q^1_e and q^2_s denote the resulting and starting set of q^1 and q^2, respectively. Furthermore, let $Q^3 = Q^1 \wedge Q^2$. If $q^1_e \cap q^2_s \neq \varnothing$, then $Q^3(D)$ is the union of vertices in paths in $Q^1(D) \cup Q^2(D)$ which pass through $q^1_e \cup q^2_s$, i.e., the set of queries for which the starting and the ending points are known, is closed under conjunction.*

Let q be a regular path query corresponding to a regular expression r_q, where the latter is over an alphabet of edge symbols E. Since regular languages are closed under the operation of reversal, $reverse(r_q)$ is also a regular expression which corresponds to query $q^{reverse}$ in a way that $reverse(r_q)$ is isomorphic to $q^{reverse}$.

Let I be an instance over $\mathbf{S}$. Let us define mappings f from the strings generated by r_q to all paths in I. Similarly, let f^r be a mapping from strings

[5] An OMQ $Q = (\mathbf{S}, \Sigma, q)$ is dyadic if the maximum arity of all predicates in $\mathbf{S}$, Σ and q is 2.

generated by $reverse(r_q)$ to all paths in I. Since the edges of q and $q^{reverse}$ are the same, it can be concluded that $f = f^r$. Based on the above argument, it is clear that $q(I) = q^{reverse}(I)$, i.e., any query is contained in its reverse.[6]

Given a query $Q = (\mathbf{S}, \Sigma, q) \in \mathsf{PG2RQ}^k$, for each component q_i of q, let $Q_i = (\mathbf{S}, \Sigma, q_i)$ be the OMQ which corresponds to q_i. If Q is distributable, then based on Lemma 80, there is some Q_i which is contained in Q. Moreover, as a corollary to Lemma 88, if Q_i is reversible, then (i) Q_i is contained in $Q_i^{reverse}$, i.e., $Q_i \subseteq Q_i^{reverse}$, and (ii) if Q' is a reversible query, then $Q'' = Q_i \wedge Q'$ is also reversible, and $Q'' \subseteq Q''^{reverse}$.

In our test corpora we evaluate the performance of query answering for distributable queries with/without reversal. It was observed that in many cases, evaluation of $Q_i^{reverse}$, i.e., the reversal of the ith component of Q (Q_i), is easier than evaluation of Q_i itself. Furthermore, similar to [68] we conducted redundancy analysis for the ontologies in our collection in which we utilized the results of Lemma 88. Before we present our results on this analysis, let us recall the notion of redundancy from [68] which is applied to $\mathsf{PG2RQ}^k$ queries in Definition 89.

Definition 89. [68] Let $Q = (\mathbf{S}, \Sigma, q) \in \mathsf{PG2RQ}^k$, where $q : V_s \to V_e$ in which V_s is a known $\mathbf{S}$-database and V_e is a known answer set. The *redundancy* of Q is defined as the number of atoms which are reachable from nodes of V_s for which there is no path to V_e unless it requires going back along an edge that is already traversed.

Another optimization which we utilize is minimization of redundancy, which are intuitively the number of dead-ended paths from the source. In order to evaluate the direction of edges to traverse, we need to choose a direction with the lowest redundancy. In our collection, we came across some redundancies in answering some queries such as the following.

From OBO: An ontology describing amphibian taxonomy consists of part_of(x, y) predicate which evaluates to true if x is a part of y. For instance,

[6]Note that here we do not consider the case of folded queries, where the conjunction of queries with their inverse may collapse into identity edges.

x can be lamina anterior of maxilla that is the anterior part of the maxilla which is in contact with the premaxilla[7] which is part of maxilla.[8]

It is known that part_of is a transitive predicate. It can be seen that part_of may connect different parts of the body. Each part of the body may be part of multiple body parts, and at the same time, there may be multiple y for each part of the body x such that part_of(x, y) holds true. Therefore, part_of is a many-to-many mapping in general. In our collection, we discovered that for each x there are far more y such that part_of(x, y) holds true, than there are x for each y to satisfy part_of(x, y).

According to what argued above, we can define the inverse of part_of(x, y) as has_part(y, x), and whenever the given query is reversible and only consists of part_of predicates, we prefer has_part to part_of, as the former possesses a lower redundancy as defined in Definition 89. Furthermore, as a corollary to Lemma 88, if a query is a conjunction of two reversible queries, it is also reversible and the above optimization to minimize redundancy can be applied. For example, we may query all amphibian body parts which are part of upper jaw while being located posterior to maxilla, and the answer will not include premaxilla as it is located anterior to maxilla in dorsal view.[9] It can be seen that the above query is the conjunction of part_of and posterior_to queries, both of which are reversible. This sets it in a position to be optimized by computing its reverse which is the conjunction of has_part and anterior_to queries. The results of this optimization are summarized in Table 7.4. Notice that the numbers of this table include the reverse of conjunctive queries as Lemma 88 suggests.

Exploiting parallelism in answering acyclic queries

Recall that the data complexity of query evaluation of GRQ and therefore PG2RQ^k belongs to NLogSpace, and therefore it is highly parallelizable. In general, a problem is called highly parallelizable if it can be solved in polylogarithmic time utilizing a polynomial number of processors that are exploited

[7] $http://purl.org/obo/owl/AAO\#AAO\_0000651 > lamina\_anterior\_of\_maxilla$.

[8] $http://purl.org/obo/owl/AAO\#AAO\_0000285 > maxilla$, defined as paired, intramembranous bone located on the lateral sides of the skull, posterior to the premaxillae.

[9] Derived from Latin dorsum which means 'back'.

Table 7.4: Optimization results for query answering (Graal)

# Evaluated queries	**# Reversible queries**	**Avg. QA time (Direct) (s)**	**Avg. QA time (Reverse) (s)**
62	41	71.0	32.8

in parallel [90].

Note that the computation of acyclic conjunctive queries (ACQs) is also highly parallelizable. An parallel algorithm for evaluation of ACQs is proposed in Section 7.3 of [79]. Therefore, the evaluation of OMQs that result from combining GRD rule sets and ACQs can be further optimized using the high parallelizability inherent in ACQs. We have not conducted experiments for this combination as it requires a new algorithm for optimizing these queries which is beyond the focus of the current work and is left as a future work.

For the future work, we can also consider the effect of increasing k on the performance of distribution checking over components and on the query evaluation.

Chapter 8

Conclusion and Future Directions

8.1 Restricted Chase Termination

In the first part of the contributions of this dissertation (i.e., Chapters 4 and 5), we introduced a general framework to extend classes of chase terminating rule sets. We formulated a technique to characterize finite restricted chase which can be applied to extend any class of finite Skolem chase identified by a condition of acyclicity. The main strength of our work, which is also the main distinction from almost all previous work on chase termination, is its generality. Then, we showed how to apply our techniques to extend δ-bounded rule sets. Our theoretical results for complexity analyses showed that in general this extension indeed increases the complexities of membership checking and the complexity of combined reasoning tasks for δ-bounded rule sets under the restricted chase compared to the Skolem chase. However, by implementation and experimentation, we showed that this increase does not prohibit the applicability of our work in real-world ontologies. Our experimental results discovered a growing number of practical ontologies with finite restricted chase obtained by increasing the length of cycles as well as changing the underlying cycle function. They also showed evidence that existential rules provide a suitable modelling language for ontological reasoning.

8.2 Distribution over Components for Ontology-Mediated Query Answering

The topic of chapters 6 and 7 is related to the problem of distributed query answering of incomplete data in the presence of ontologies. This is indispensable in applications where the data is heterogeneous and comes from different sources in massive volumes. This problem asks whether it is possible to distribute the query workload among different machines, i.e., whether we can compute the answer to a database query in the presence of an ontology by parallelizing it over the connected components of the database.

Only if the answer to the above problem is positive for a given ontology-mediated query belonging to a particular class, then we can compute the query in a coordination-free and distributed manner. From the literature, it is known that this problem is in general undecidable when the database query belongs to the class of conjunctive queries and the ontology belongs to that of Datalog.

In chapters 6 and 7, we present our contributions on the above problem for two main classes of ontology-mediated queries. The first class, that is the topic of Chapter 6, is constructed from disjunctive tuple-generating dependencies and conjunctive queries. In the second class which is presented in Chapter 7, we study distributed reasoning for the class of ontology-mediated queries formed from generalized regular queries and conjunctive queries.

The above classes of ontology-mediated queries involve useful language constructs for navigation of (hyper)graph databases as well as disjunction with which nondeterminism can be modelled. However, to the best of our knowledge, this is the first time the problem of distributed reasoning is theoretically studied for these two classes of ontology-mediated queries.

Additionally, we are not aware of any empirical analysis of any kind to evaluate the practical benefits of distribution over connected components of databases. Therefore, our experiments to evaluate the performance gain of reasoning with distributable ontology-mediated queries for the classes of linear and linear disjunctive tuple-generating dependencies are the first of its kind.

8.2.1 Distribution over Components for Restricted Weakly Linear Queries

In Chapter 6, we studied distributed reasoning for a class of disjunctive TGD we introduced as restricted weakly-linear disjunctive tuple-generating dependencies. We discovered sufficient conditions for OMQs based on restricted weakly-linear disjunctive tuple-generating dependencies to be distributable over connected database components. In particular, for the OMQs based on these tuple-generating dependencies with acyclic and quantifier-free conjunctive queries we characterized a subclass called bidirectionally-guarded that has a decidable problem of distribution over components, and for which one can characterize distributable fragments.

Lifting the guardedness property of the input ontology, we further identified some fragments of the resulting ontologies combined with answer-guarded CQs which yield decidable distribution problem. Our analyses demonstrate that for this problem the complexity upper bounds ranging from ExpTime to 2ExpTime can be obtained when applied to these fragments.

The experiments demonstrate that query answering computation times computed using state of the art chase engines can be significantly improved in selected ontology benchmarks for distributable ontology-mediated queries. The results of this chapter can benefit chase-based and rewriting-based query answering approaches alike.

8.2.2 Distribution over Components for Generalized Regular Datalog

In Chapter 7, we studied the same problem of distribution over components considered in Chapter 6 by focusing on generalized regular Datalog ontology-mediated query answering. We introduced sufficient conditions for ontology-mediated queries based on generalized regular Datalog to be distributable over connected database components.

As opposed to the case of RWL ontology-mediated queries, the results of this chapter shows that we can semantically characterize the fragment of

these queries that distributes over components uniformly for the whole fragment. Moreover, we identified a syntactic subset of these queries called pseudo-guarded generalized regular queries (PG2RQ) that restricts generalized regular Datalog rules to be syntactically connected, where each component has a guard atom for which the above problem is shown to be decidable. Our complexity results for deciding distribution over connected components for PG2RQ queries yields an upper bound of 2ExpSpace.

The latter result has been extended to uniformly cover a wide range of query languages known in the literature, along with their combination with transitive closure using the same techniques developed therein.

Our experiments evaluate the query answering performance gain achieved by focusing on distributable PG2RQ ontology-mediated queries along with different optimizations performed to improve query answering. Notice that similar to the previous chapter, the results of this chapter benefits non-chase based algorithms for query answering as well.

8.3 Future work

In this section we list a number of possible directions of future research which can be pursued from the work we conducted in this thesis and beyond.

8.3.1 Restricted Chase Termination

To speed up membership checking of k-safe rules, it is an interesting open problem whether there exist conditions for subclasses of these rule sets while reducing the cost of checking. One idea is to investigate syntactic conditions under which triggers to a rule are necessarily active. The current implementation of our system is relatively slow, particularly for nonlinear rules. It often requires long chase times to check membership in k-safe(Φ) even for small values of k. In order to tackle this problem, we can consider two options which can also be combined toward a more efficient implementation. The first option is considering k-safe(Φ) under a particular path selection strategy such as the Datalog-first approach. This way, we can filter out a subset of paths in our

membership analysis.

Additionally, adding negative constraints (rules with empty head) and negations under different semantics to the body of rules presently poses challenge to the reasoning/termination of the chase. An open problem is how adding the above constructs generally affect reasoning/chase termination for different variants of the chase. In particular, in [83], query answering under the stable model semantics has been studied for guarded existential rules with negation in which rule bodies may contain negated atoms.

More recently, in [7], the stable model semantics has been considered for *normal tuple-generating dependencies* (NTGDs), which are existential rules enriched with default negation (also known as negation as failure). In this regard, the query answering is shown to be decidable for the class of weakly-acyclic NTGDs. However, the same problem is shown to be undecidable for sticky and guarded NTGDs.

Given the power of the restricted chase recently obtained in [104] which show that terminating restricted chase can achieve nonelementary data complexity bounds, it is an interesting open problem to study stable model semantics for different classes of terminating NTGDs under the restricted chase and to see how reasoning/termination is affected in this setting.

8.3.2 Distribution over Components

Recently, a number of FO-rewritable fragments for existential rules have been characterized [23]. For the future work, it is worthwhile to study distributability over components for more expressive languages such as (frontier-) guarded (disjunctive) rules. Also another challenging problem is distributability of linear disjunctive TGDs in the presence of (the more general) conjunctive queries.

In Chapter 6, we found a way to handle disjunctive TGDs for our purpose using the transformation of a fragment of disjunctive TGDs which is Datalog-reducible. However, a general reasoner that can handle these rules is currently missing in the literature. Such a reasoner can be specifically optimized for query answering with these rules.

Moreover, we can extend our experiments to check distribution over compo-

nents with bidirectionally-guarded TGDs and answer-guarded acyclic (as well as quantifier-free) CQs. As shown in Section 6.4, the theoretical complexity upper bound for this check will be higher than that of singly bidirectionally-guarded considered in our experiments of Section 6.7. However, it is an interesting problem to evaluate the expressiveness/complexity/practicality balance in this setting.

In Section 7.4 of Chapter 7, we showed a concrete algorithm to decide distribution over components for GRQ queries. The algorithm presented over there was based on applying the connecting operator on these queries to obtain another query on which we can mount our equivalence checking algorithm that is the basis of our implementation. Given that the class of these ontology-mediated queries is more expressive, as a future work, one can focus on these queries and evaluate the performance of distributability checking with them. This may have substantial implications in big graph database applications where this language is known to be a suitable for expressive query evaluation involving graph paths.

Besides, the language of Monadic Datalog (MDL) shown in Table 7.1 is considered as a primary language for web information extraction (wrapping) in [77]. Due to the limited expressiveness of this language, and as a substitute for it, one can consider the language of GRQ queries with potentials in the task of automated web data extraction. Our results on the distribution over components using these two fragments may improve the performance of data extraction or be integrated into a domain-specific language (DSL) which may involve sophisticated querying mechanisms.

References

[1] Serge Abiteboul, Peter Buneman, and Dan Suciu. *Data on the Web: from relations to semistructured data and XML*. Morgan Kaufmann, 2000.

[2] Serge Abiteboul, Richard Hull, and Victor Vianu. *Foundations of databases: the logical level*. Addison-Wesley Longman Publishing Co., Inc., 1995.

[3] Serge Abiteboul and Victor Vianu. "Regular path queries with constraints". In: *Journal of Computer and System Sciences* 58.3 (1999), pp. 428–452.

[4] Foto N Afrati and Phokion G Kolaitis. "Repair checking in inconsistent databases: algorithms and complexity". In: *Proceedings of the Twelvth International Conference on Database Theory*. ACM. 2009, pp. 31–41.

[5] Shqiponja Ahmetaj, Magdalena Ortiz, and Mantas Simkus. "Rewriting guarded existential rules into small datalog programs". In: *Proceedings of the Twenty-First International Conference on Database Theory*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik. 2018.

[6] Alfred V Aho, Catriel Beeri, and Jeffrey D Ullman. "The theory of joins in relational databases". In: *ACM Transactions on Database Systems (TODS)* 4.3 (1979), pp. 297–314.

[7] Mario Alviano, Michael Morak, and Andreas Pieris. "Stable model semantics for tuple-generating dependencies revisited". In: *Proceedings of the 36th ACM SIGMOD Symposium on Principles of Database Systems*. 2017, pp. 377–388.

[8] Tom J Ameloot et al. "Datalog queries distributing over components". In: *ACM Transactions on Computational Logic (TOCL)* 18.1 (2017), p. 5.

[9] Hajnal Andréka, István Németi, and Johan van Benthem. "Modal languages and bounded fragments of predicate logic". In: *Journal of Philosophical Logic* 27.3 (1998), pp. 217–274.

[10] Marcelo Arenas, Leopoldo Bertossi, and Jan Chomicki. "Consistent query answers in inconsistent databases". In: *Proceedings of the Eighteenth ACM Symposium on Principles of Database Systems*. ACM. 1999, pp. 68–79.

[11] Alessandro Artale et al. "The DL-Lite family and relations". In: *Journal of Artificial Intelligence Research* 36 (2009), pp. 1–69.

[12] Jean-François Baget. "Improving the forward chaining algorithm for conceptual graphs rules." In: *Proceedings of the Ninth International Conference on Principles of Knowledge Representation and Reasoning*. AAAI Press, 2004, pp. 407–414.

[13] Jean-François Baget et al. "Combining existential rules and transitivity: Next steps". In: *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence*. 2015.

[14] Jean-François Baget et al. "Datalog+, RuleML and OWL 2: Formats and translations for existential rules." In: *Proceedings of the RuleML 2015 Challenge (Challenge+ DC@ RuleML)*. Vol. 1417. CEUR Workshop Proceedings. CEUR-WS.org, 2015.

[15] Jean-François Baget et al. "Extending acyclicity notions for existential rules." In: *Proceedings of the 21st European Conference on Artificial Intelligence*. IOS Press, 2014, pp. 39–44.

[16] Jean-François Baget et al. "Extending decidable cases for rules with existential variables". In: *Proceedings of the Twenty-First International Joint Conference on Artificial Intelligence*. 2009, pp. 677–682.

[17] Jean-François Baget et al. "Graal: A toolkit for query answering with existential rules". In: *Proceedings of the International Symposium on Rules and Rule Markup Languages for the Semantic Web*. Vol. 9202. LNCS. Springer. 2015, pp. 328–344.

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

[19] Jean-François Baget et al. "Revisiting chase termination for existential rules and their extension to nonmonotonic negation". In: *Proceedings of the 15th International Workshop on Non-Monotonic Reasoning (NMR 2014)*. 2014.

[20] Vince Bárány, Balder Ten Cate, and Luc Segoufin. "Guarded negation". In: *Journal of the ACM* 62.3 (2015), p. 22.

[21] Vince Bárány, Balder Ten Cate, and Martin Otto. "Queries with guarded negation". In: *Proceedings of the VLDB Endowment* 5.11 (2012), pp. 1328–1339.

[22] Vince Bárány, Balder Ten Cate, and Luc Segoufin. "Guarded negation". In: *Proceedings of the International Colloquium on Automata, Languages, and Programming*. Springer. 2011, pp. 356–367.

[23] Pablo Barceló et al. "First-order rewritability of frontier-guarded ontology-mediated queries". In: *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence*. AAAI Press. 2018, pp. 1707–1713.

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

[25] Catriel Beeri and Moshe Y Vardi. "The implication problem for data dependencies". In: *Proceedings of the 8th International Colloquium on Automata, Languages, and Programming*. Vol. 115. LNCS. Springer. 1981, pp. 73–85.

[26] Luigi Bellomarini, Emanuel Sallinger, and Georg Gottlob. "The Vadalog system: Datalog-based reasoning for knowledge graphs". In: *Proceedings of the VLDB Endowment* 11.9 (2018), pp. 975–987.

[27] Michael Benedikt. "How can reasoners simplify database querying (and why haven't they done it yet)?" In: *Proceedings of the 37th ACM Symposium on Principles of Database Systems*. ACM. 2018, pp. 1–15.

[28] Michael Benedikt, Pierre Bourhis, and Michael Vanden Boom. "A step up in expressiveness of decidable fixpoint logics". In: *Proceedings of the 31st Annual ACM/IEEE Symposium on Logic in Computer Science*. IEEE. 2016, pp. 1–10.

[29] Michael Benedikt, Pierre Bourhis, and Pierre Senellart. "Monadic datalog containment". In: *Proceedings of the International Colloquium on Automata, Languages, and Programming*. Springer. 2012, pp. 79–91.

[30] Michael Benedikt, Julien Leblay, and Efthymia Tsamoura. "PDQ: Proof-driven query answering over web-based data". In: *Proceedings of the VLDB Endowment* 7.13 (2014), pp. 1553–1556.

[31] Michael Benedikt et al. "Benchmarking the chase". In: *Proceedings of the 36th ACM Symposium on Principles of Database Systems*. ACM, 2017, pp. 37–52.

[32] Gerald Berger and Andreas Pieris. "Ontology-mediated queries distributing over components". In: *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence*. 2016, pp. 943–949.

[33] 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), p. 33.

[34] Angela Bonifati and Stefania Dumbrava. "Graph queries: From theory to practice". In: *ACM SIGMOD Record* 47.4 (2019), pp. 5–16.

[35] Angela Bonifati, Stefania Dumbrava, and Emilio Jesús Gallego Arias. "Certified graph view maintenance with regular datalog". In: *Theory and Practice of Logic Programming* 18.3-4 (2018), pp. 372–389.

[36] Pierre Bourhis, Markus Krötzsch, and Sebastian Rudolph. "How to best nest regular path queries". In: *Informal Proceedings of the Twenty-Seventh International Workshop on Description Logics*. 2014.

[37] Pierre Bourhis, Markus Krötzsch, and Sebastian Rudolph. "Reasonable highly expressive query languages". In: *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence*. 2015.

[38] Pierre Bourhis et al. "Guarded-based disjunctive tuple-generating dependencies". In: *ACM Transactions on Database Systems* 41.4 (2016), 27:1–27:45.

[39] Peter Buneman. "Semistructured Data". In: *Proceedings of the Sixteenth ACM Symposium on Principles of Database Systems*. PODS '97. Tucson, Arizona, USA: ACM, 1997, pp. 117–121. ISBN: 0-89791-910-6. DOI: 10.1145/263661.263675. URL: http://doi.acm.org/10.1145/263661.263675.

[40] Marco Calautti, Georg Gottlob, and Andreas Pieris. "Chase termination for guarded existential rules". In: *Proceedings of the 34th ACM Symposium on Principles of Database Systems*. ACM. 2015, pp. 91–103.

[41] Marco Calautti and Andreas Pieris. "Oblivious chase termination: The sticky case". In: *Proceedings of the Twenty-Second International Conference on Database Theory*. Vol. 127. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019, 17:1–17:18.

[42] Andrea Calì, Georg Gottlob, and Thomas Lukasiewicz. "A general datalog-based framework for tractable query answering over ontologies". In: *Web Semantics: Science, Services and Agents on the World Wide Web* 14 (2012), pp. 57–83.

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

[44] Andrea Calì et al. "Data integration under integrity constraints". In: *Proceedings of the Seminal Contributions to Information Systems Engineering*. Springer, 2013, pp. 335–352.

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

[46] Diego Calvanese, Giuseppe De Giacomo, and Moshe Y Vardi. "Decidable containment of recursive queries". In: *Theoretical Computer Science* 336.1 (2005), pp. 33–56.

[47] Diego Calvanese et al. “Query processing using views for regular path queries with inverse”. In: *Proceedings of the Nineteenth ACM Symposium on Principles of Database Systems (PODS 2000)*. 2000, pp. 58–66.

[48] Diego Calvanese et al. “Tractable reasoning and efficient query answering in description logics: The DL-Lite family”. In: *Journal of Automated Reasoning* 39.3 (2007), pp. 385–429.

[49] David Carral, Irina Dragoste, and Markus Krötzsch. “Restricted chase (non) termination for existential rules with disjunctions”. In: *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence*. ijcai.org, 2017, pp. 922–928.

[50] David Carral et al. “EL-ifying Ontologies”. In: *Proceedings of the 7th International Joint Conference on Automated Reasoning*. Vol. 8562. LNCS. Springer. 2014, pp. 464–479.

[51] David Carral et al. “VLog: A rule engine for knowledge graphs”. In: *Proceedings of the16th International Semantic Web Conference*. Vol. 11779. LNCS. Springer. 2019, pp. 19–35.

[52] 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.

[53] Michelle Cheatham and Catia Pesquita. “Semantic Data Integration”. In: *Handbook of Big Data Technologies*. Springer, 2017, pp. 263–305.

[54] *Concept Base.cc: A system for metamodeling and method engineering*. `http://conceptbase.sourceforge.net`. Accessed: 2016-11-24.

[55] Mariano P Consens and Alberto O Mendelzon. “GraphLog: a visual formalism for real life recursion”. In: *Proceedings of the ninth ACM SIGMOD Symposium on Principles of Database Systems*. 1990, pp. 404–416.

[56] Bernardo 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.

[57] Ivan Damgård et al. “Multiparty computation from somewhat homomorphic encryption”. In: *Proceedings of the Annual Cryptology Conference*. Springer. 2012, pp. 643–662.

[58] Daniel Danielski and Emanuel Kieronski. “Finite satisfiability of unary negation fragment with transitivity”. In: *arXiv preprint arXiv:1809.03245* (2018).

[59] Evgeny Dantsin et al. “Complexity and expressive power of logic programming”. In: *ACM Computing Surveys* 33.3 (2001), pp. 374–425.

[60] Stathis Delivorias et al. “On the k-boundedness for existential rules”. In: *Proceedings of the 2nd International Joint Conference on Rules and Reasoning*. Vol. 11092. LNCS. Springer. 2018, pp. 48–64.

[61] Alin Deutsch, Alan Nash, and Jeff Remmel. “The chase revisited”. In: *Proceedings of the Twenty-Seventh ACM Symposium on Principles of Database Systems*. ACM. 2008, pp. 149–158.

[62] *DLV system: A system for metamodeling and method engineering*. `http://www.dlvsystem.com`. Accessed: 2016-11-24.

[63] Thomas Eiter, Georg Gottlob, and Heikki Mannila. “Disjunctive datalog”. In: *ACM Transactions on Database Systems (TODS)* 22.3 (1997), pp. 364–418.

[64] Ronald Fagin et al. “Clio: Schema mapping creation and data exchange”. In: *Proceedings of the Conceptual Modeling: Foundations and Applications*. Springer, 2009, pp. 198–236.

[65] Ronald Fagin et al. “Data exchange: Semantics and query answering”. In: *Proceedings of the 9th International Conference on Database Theory*. Vol. 2572. LNCS. Springer. 2003, pp. 207–224.

[66] Ronald Fagin et al. “Data exchange: Semantics and query answering”. In: *Theoretical Computer Science* 336.1 (2005), pp. 89–124.

[67] Daniela Florescu, Alon Levy, and Dan Suciu. “Query containment for conjunctive queries with regular expressions”. In: *Proceedings of the Seventeenth ACM Symposium on Principles of Database Systems*. 1998, pp. 139–148.

[68] Darius Foo et al. “SGL: A domain-specific language for large-scale analysis of open-source code”. In: *Proceedings of the IEEE Cybersecurity Development (SecDev)*. IEEE. 2018, pp. 61–68.

[69] Ariel Fuxman et al. “Peer data exchange”. In: *ACM Transactions on Database Systems (TODS)* 31.4 (2006), pp. 1454–1498.

[70] Herve Gallaire, Jack Minker, and Jean-Marie Nicolas. “Logic and databases: A deductive approach”. In: *ACM Computing Surveys* 16.2 (1984), pp. 153–185.

[71] Tom Gardiner, Dmitry Tsarkov, and Ian Horrocks. “Framework for an automated comparison of description logic reasoners”. In: *Proceedings of the International Semantic Web Conference*. Springer. 2006, pp. 654–667.

[72] Birte Glimm, Yevgeny Kazakov, and Trung-Kien Tran. “Ontology materialization by abstraction refinement in horn SHOIF”. In: *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence*. 2017.

[73] Birte Glimm et al. “Abstraction refinement for ontology materialization”. In: *Proceedings of the International Semantic Web Conference*. Springer. 2014, pp. 180–195.

[74] Tomasz Gogacz and Jerzy Marcinkowski. "All–instances termination of chase is undecidable". In: *Proceedings of the International Colloquium on Automata, Languages, and Programming*. Springer. 2014, pp. 293–304.

[75] Tomasz Gogacz, Jerzy Marcinkowski, and Andreas Pieris. "All-instances restricted Chase termination: The guarded case". In: *arXiv preprint arXiv:1901.03897* (2019).

[76] Oded Goldreich, Silvio Micali, and Avi Wigderson. "How to play any mental game". In: *Proceedings of the Nineteenth Annual ACM Symposium on Theory of Computing*. ACM. 1987, pp. 218–229.

[77] Georg Gottlob and Christoph Koch. "Monadic datalog and the expressive power of languages for web information extraction". In: *Journal of the ACM* 51.1 (2004), pp. 74–113.

[78] Georg Gottlob, Nicola Leone, and Francesco Scarcello. "Robbers, marshals, and guards: game theoretic and logical characterizations of hypertree width". In: *Journal of Computer and System Sciences* 66.4 (2003), pp. 775–808.

[79] Georg Gottlob, Nicola Leone, and Francesco Scarcello. "The complexity of acyclic conjunctive queries". In: *Journal of the ACM* 48.3 (2001), pp. 431–498.

[80] Georg Gottlob, Giorgio Orsi, and Andreas Pieris. "Ontological queries: Rewriting and optimization". In: *Proceedings of the IEEE Twenty-Seventh International Conference on Data Engineering*. IEEE. 2011, pp. 2–13.

[81] Georg Gottlob, Sebastian Rudolph, and Mantas Simkus. "Expressiveness of guarded existential rule languages". In: *Proceedings of the Thirty-Third ACM Symposium on Principles of Database Systems*. ACM. 2014, pp. 27–38.

[82] Georg Gottlob et al. "On the complexity of ontological reasoning under disjunctive existential rules". In: *Proceedings of the International Symposium on Mathematical Foundations of Computer Science*. Springer. 2012, pp. 1–18.

[83] Georg Gottlob et al. "Stable model semantics for guarded existential rules and description logics". In: *Proceedings of the Fourteenth International Conference on the Principles of Knowledge Representation and Reasoning*. 2014.

[84] Erich Grädel. "On the restraining power of guards". In: *The Journal of Symbolic Logic* 64.4 (1999), pp. 1719–1742.

[85] Gösta Grahne and Adrian Onet. "Anatomy of the chase". In: *Fundamenta Informaticae* 157.3 (2018), pp. 221–270. DOI: 10.3233/FI-2018-1627. URL: https://doi.org/10.3233/FI-2018-1627.

[86] Gösta Grahne and Adrian Onet. "Data correspondence, exchange and repair". In: *Proceedings of the Thirteenth International Conference on Database Theory*. ACM. 2010, pp. 219–230.

[87] Gosta Grahne and Adrian Onet. "The data-exchange chase under the microscope". In: *arXiv preprint arXiv:1407.2279* (2014).

[88] 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.

[89] Bernardo Cuenca Grau et al. "Computing datalog rewritings beyond horn ontologies". In: *Proceedings of the Twenty-Third International Joint Conference on Artificial Intelligence*. 2013.

[90] Raymond Greenlaw, H James Hoover, Walter L Ruzzo, et al. *Limits to parallel computation: P-completeness theory*. Oxford University Press on Demand, 1995.

[91] Pavol Hell and Jaroslav Nešetřil. "The core of a graph". In: *Discrete Mathematics* 109.1-3 (1992), pp. 117–126.

[92] Gerd G Hillebrand et al. "Undecidable boundedness problems for datalog programs". In: *Journal of Logic Programming* 25.2 (1995), pp. 163–190.

[93] Matthew Horridge and Sean Bechhofer. "The OWL API: A java API for OWL ontologies". In: *Semantic Web* 2.1 (2011), pp. 11–21.

[94] Yin-Fu Huang and Wei-Cheng Chen. "Parallel query on the in-memory database in a CUDA platform". In: *Proceedings of the 10th International Conference on P2P, Parallel, Grid, Cloud and Internet Computing (3PGCIC)*. IEEE. 2015, pp. 236–243.

[95] Neil Immerman. "Nondeterministic space is closed under complementation". In: *SIAM Journal on Computing* 17.5 (1988), pp. 935–938.

[96] 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.

[97] Mark Kaminski, Yavor Nenov, and Bernardo Cuenca Grau. "Datalog rewritability of disjunctive datalog programs and its applications to ontology reasoning". In: *Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence*. AAAI Press. 2014, pp. 1077–1083.

[98] Mark Kaminski, Yavor Nenov, and Bernardo Cuenca Grau. "Datalog rewritability of Disjunctive Datalog programs and non-Horn ontologies". In: *Artificial Intelligence* 236 (2016), pp. 90–118.

[99] Arash Karimi, Heng Zhang, and Jia-Huai You. "Beyond skolem chase: A study of finite chase under standard chase variant". In: *Proceedings of the 30th International Workshop on Description Logics, Montpellier, France, July 18-21, 2017*. Vol. 1879. CEUR-WS.org, 2017.

[100] Arash Karimi, Heng Zhang, and Jia-Huai You. "Restricted chase termination: A hierarchical approach and experimentation". In: *Proceedings of the 2nd International Joint Conference on Rules and Reasoning*. Vol. 11092. LNCS. Springer. 2018, pp. 98–114.

[101] Arash Karimi, Heng Zhang, and Jia-Huai You. "Restricted chase termination for existential rules: A hierarchical approach and experimentation". In: *CoRR* abs/2005.05423 (2020). arXiv: 2005.05423. URL: https://arxiv.org/abs/2005.05423.

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

[103] Markus Krötzsch. "Too much information: Can AI cope with modern knowledge graphs?" In: *Proceedings of the International Conference on Formal Concept Analysis*. Springer. 2019, pp. 17–31.

[104] Markus Krötzsch, Maximilian Marx, and Sebastian Rudolph. "The power of the terminating chase". In: *Proceedings of the Twenty-Second International Conference on Database Theory*. Vol. 127. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019, 3:1–3:17.

[105] Markus Krötzsch and Sebastian Rudolph. "Extending decidable existential rules by joining acyclicity and guardedness". In: *Proceedings of the Twenty-Second International Joint Conference on Artificial Intelligence*. IJCAI/AAAI, 2011, pp. 963–968.

[106] Michel Leclère et al. "A single approach to decide chase termination on linear existential rules". In: *Proceedings of the Twenty-Second International Conference on Database Theory*. Vol. 127. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019, 18:1–18:19.

[107] Jens Lehmann et al. "DBpedia–a large-scale, multilingual knowledge base extracted from Wikipedia". In: *Semantic Web* 6.2 (2015), pp. 167–195.

[108] Maurizio Lenzerini. "Data integration: A theoretical perspective". In: *Proceedings of the Twenty-First ACM Symposium on Principles of Database Systems*. ACM. 2002, pp. 233–246.

[109] Carsten Lutz and Frank Wolter. "Non-uniform data complexity of query answering in description logics". In: *Proceedings of the Thirteenth International Conference on the Principles of Knowledge Representation and Reasoning*. 2012.

[110] Carsten Lutz et al. "The combined approach to OBDA: Taming role hierarchies using filters". In: *Proceedings of the International Semantic Web Conference*. Springer. 2013, pp. 314–330.

[111] 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.

[112] Jerzy Marcinkowski. "Achilles, turtle, and undecidable boundedness problems for small datalog programs". In: *SIAM Journal on Computing* 29.1 (1999), pp. 231–257.

[113] Bruno Marnette. "Generalized schema-mappings: from termination to tractability". In: *Proceedings of the Twenty-Eighth ACM Symposium on Principles of Database Systems*. ACM, 2009, pp. 13–22.

[114] Maximilian Marx, Markus Krötzsch, and Veronika Thost. "Logic on MARS: ontologies for generalised property graphs". In: *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence*. 2017, pp. 1188–1194.

[115] Nicolas Matentzoglu and Bijan Parsia. *The Manchester OWL Corpus (MOWLCorp), original serialisation*. July 2014. DOI: 10.5281/zenodo.10851. URL: https://doi.org/10.5281/zenodo.10851.

[116] Michael Meier. "The backchase revisited". In: *The VLDB Journal—The International Journal on Very Large Data Bases* 23.3 (2014), pp. 495–516.

[117] Tom Mitchell et al. "Never-ending learning". In: *Communications of the ACM* 61.5 (2018), pp. 103–115.

[118] Boris Motik. "Reasoning in description logics using resolution and deductive databases". PhD thesis. Karlsruhe Institute of Technology, Germany, 2006.

[119] Boris Motik et al. "Parallel materialisation of datalog programs in centralised, main-memory RDF systems". In: *Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence*. 2014.

[120] Yavor Nenov et al. "RDFox: A highly-scalable RDF store". In: *Proceedings of the International Semantic Web Conference*. Springer. 2015, pp. 3–20.

[121] Adrian Onet. "The chase procedure and its applications in data exchange". In: *Data Exchange, Integration, and Streams*. Vol. 5. Dagstuhl Follow-Ups. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2013, pp. 1–37.

[122] Magdalena Ortiz, Sebastian Rudolph, and Mantas Simkus. "Worst-case optimal reasoning for the Horn-DL fragments of OWL 1 and 2". In: *Proceedings of the Twelfth International Conference on the Principles of Knowledge Representation and Reasoning*. 2010.

[123] Christos H. Papadimitriou. *Computational complexity*. Addison-Wesley, 1994, pp. I–XV, 1–523. ISBN: 978-0-201-53082-7.

[124] Antonella Poggi et al. "Linking data to ontologies". In: *Journal on Data Semantics X*. Springer, 2008, pp. 133–173.

[125] Omer Reingold. "Undirected connectivity in log-space". In: *Journal of the ACM* 55.4 (2008), p. 17.

[126] Juan L Reutter, Miguel Romero, and Moshe Y Vardi. "Regular queries on graph databases". In: *Theory of Computing Systems* 61.1 (2017), pp. 31–83.

[127] *REWERSE: Reasoning on the Web with Rules and Semantics*. `http://rewerse.net`. Accessed: 2016-11-24.

[128] Sebastian Rudolph and Michaël Thomazo. "Characterization of the expressivity of existential rule queries". In: *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence*. 2015, pp. 3193–3199.

[129] Sebastian Rudolph and Michaël Thomazo. "Expressivity of datalog variants–completing the picture". In: *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence*. 2016.

[130] Vladislav Rutenburg. "Complexity of generalized graph coloring". In: *Proceedings of the International Symposium on Mathematical Foundations of Computer Science*. Vol. 233. LNCS. Springer. 1986, pp. 573–581.

[131] Walter J Savitch. "Relationships between nondeterministic and deterministic tape complexities". In: *Journal of Computer and System Sciences* 4.2 (1970), pp. 177–192.

[132] Adi Shamir. "How to share a secret". In: *Communications of the ACM* 22.11 (1979), pp. 612–613.

[133] Oded Shmueli. "Equivalence of datalog queries is undecidable". In: *The Journal of Logic Programming* 15.3 (1993), pp. 231–241.

[134] Barry Smith et al. "The OBO Foundry: coordinated evolution of ontologies to support biomedical data integration". In: *Nature biotechnology* 25.11 (2007), p. 1251.

[135] Fabian M Suchanek, Gjergji Kasneci, and Gerhard Weikum. "Yago: a core of semantic knowledge". In: *Proceedings of the 16th International Conference on World Wide Web*. ACM. 2007, pp. 697–706.

[136] Balder Ten Cate and Luc Segoufin. "Unary negation". In: *Logical Methods in Computer Science* 9.3 (2013).

[137] Jacopo Urbani et al. "Efficient model construction for horn logic with VLog". In: *Proceedings of the International Joint Conference on Automated Reasoning*. Springer. 2018, pp. 680–688.

[138] Jacopo Urbani et al. "WebPIE: A web-scale parallel inference engine using MapReduce". In: *Web Semantics: Science, Services and Agents on the World Wide Web* 10 (2012), pp. 59–75.

[139] Moshe Y Vardi. "A theory of regular queries". In: *Proceedings of the Thirty-Fifth ACM Symposium on Principles of Database Systems*. ACM. 2016, pp. 1–9.

[140] Moshe Y Vardi. "Inferring multivalued dependencies from functional and join dependencies". In: *Acta Informatica* 19.4 (1983), pp. 305–324.

[141] Moshe Y Vardi. "On the complexity of bounded-variable queries". In: *Proceedings of the Fourteenth ACM Symposium on Principles of Database Systems*. 1995, pp. 266–276.

[142] Moshe Y Vardi. "The complexity of relational query languages". In: *Proceedings of the Fourteenth Annual ACM Symposium on Theory of Computing*. ACM. 1982, pp. 137–146.

[143] Denny Vrandečić and Markus Krötzsch. "Wikidata: a free collaborative knowledge base". In: (2014).

[144] Heng Zhang, Yan Zhang, and Jia-Huai You. "Existential rule languages with finite chase: Complexity and expressiveness". In: *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence*. AAAI Press, 2015, pp. 1678–1685.