TU WIEN Informatics

Foundations of Knowledge Graphs: Complexity of Arithmetic in Vadalog

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Software Engineering & Internet Computing

eingereicht von

Lucas Berent, BSc
Matrikelnummer 01625723

an der Fakultät für Informatik

der Technischen Universität Wien

Betreuung: Prof. Dr. Emanuel Sallinger
Mitwirkung: Dipl.-Ing. Markus Nissl

Wien, 30. August 2021

Lucas Berent	Emanuel Sallinger

Technische Universität Wien
A-1040 Wien ▪ Karlsplatz 13 ▪ Tel. +43-1-58801-0 ▪ www.tuwien.at

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

TU WIEN Informatics

Foundations of Knowledge Graphs: Complexity of Arithmetic in Vadalog

DIPLOMA THESIS

submitted in partial fulfillment of the requirements for the degree of

Diplom-Ingenieur

in

Software Engineering & Internet Computing

by

Lucas Berent, BSc
Registration Number 01625723

to the Faculty of Informatics

at the TU Wien

Advisor: Prof. Dr. Emanuel Sallinger
Assistance: Dipl.-Ing. Markus Nissl

Vienna, 30th August, 2021

Lucas Berent	Emanuel Sallinger

Technische Universität Wien
A-1040 Wien ▪ Karlsplatz 13 ▪ Tel. +43-1-58801-0 ▪ www.tuwien.at

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Erklärung zur Verfassung der Arbeit

Lucas Berent, BSc

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Wien, 30. August 2021

Lucas Berent

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Danksagung

Ich möchte meinem Betreuer, Emanuel Sallinger, für seine tatkräftige Unterstützung, wertvollen Einblicke und Geduld danken. Ich - sowie auch diese Arbeit - haben sehr von seinen Ideen und Diskussionen mit Emanuel profitiert. Des Weiteren möchte ich Markus Nissl meinen Dank aussprechen, der mich sehr mit strukturellen Aspekten der Arbeit, aber auch durch technische Diskussionen unterstützt hat.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Acknowledgements

First and foremost, I would like to thank my Advisor Prof. Emanuel Sallinger for his continuous support, invaluable insight, and patience. I (as well as this thesis) profited greatly through discussions and new ideas Emanuel proposed. Moreover, I am grateful for the help of Markus Nissl, who supported me greatly with structuring and proofreading the thesis, as well as with technical discussions.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Kurzfassung

Vadalog ist ein modernes Knowledge Graph (KG) System, dessen logische Kernkomponente gute (beweisbare) theoretische Garantien aufweist. Die Sprache, die in Vadalog zur Wissensrepräsentation und zur logischen Derivation verwendet wird, ist eine Erweiterung von Datalog. Die Datalog Sprache wird weitgehend - sowohl in der Forschung als auch in der Industrie - in wissensbasierten Systemen und in der Datenanalyse, im Kontext von Big Data, eingesetzt. Die zentralen Gründe für die Verwendung von Datalog sind dessen deklarative Semantik, Skalierbarkeit und die Eigenschaft, volle Rekursion ausdrücken zu können. Die mathematischen Garantien der Vadalog Sprache umfassen jedoch nicht wichtige Erweiterungen, wie etwa Arithmetik, welche in der Datenanalyse häufig benötigt werden. In der Tat gibt es keine Komplexitätsresultate für eine (entscheidbare) Datalog Sprache, erweitert durch Arithmetik und existentielle Quantifizierung in den Regelköpfen. Letztere wird unter anderem benötigt, um Ontologien logisch zu verarbeiten.

In dieser Arbeit untersuchen wir Möglichkeiten, Arithmetik in Vadalog zu integrieren und zeigen, dass es nicht trivial ist, KG Sprachen mit Arithmetik zu erweitern. Wir definieren eine neue, Logik basierte Sprache in der Form einer Erweiterung der Kernsprache von Vadalog mit einer effizienten Form von Arithmetik. Wir beweisen P-Vollständigkeit unserer Sprache, die wir *Warded Bound Datalog*$_{\mathbb{Z}}$ nennen. Des Weiteren zeigen wir die ersten Resultate der Expressivität von limit Datalog$_{\mathbb{Z}}$, welches kürzlich in einer bemerkenswerten Reihe von Publikationen von Kaminski et al. eingeführt wurde. Unsere Beiträge beweisen, dass unsere Sprache hohes Potential für die Verwendung in wissensbasierten Systemen der Künstlichen Intelligenz (KI) hat. Dadurch legen wir mit unseren theoretischen Resultaten den Grundstein für ausdrucksstarke, logische Sprachen, welche einerseits die Vielseitigkeit von komplexer, logischer Wissensableitung und Arithmetik kombinieren und andererseits effiziente Algorithmen für Applikationen in modernen KI-Systemen, wie KGs, garantieren.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Abstract

Vadalog is a powerful state-of-the-art Knowledge Graph (KG) system with strong theoretical underpinnings for its core logical reasoning component. Vadalog's reasoning and knowledge representation language is an extension of Datalog. The Datalog language is widely used in the context of knowledge-based systems and Big Data analytics because of its declarative nature, scalability, and its ability to express full recursion. The theoretical guarantees of the Vadalog language, however, do not cover extensions heavily needed in data analytics, such as arithmetic. In fact, there are no complexity results for the combination of a (decidable) Datalog language extended with arithmetic and existentials in rule heads, the latter of which is needed e.g., for ontological reasoning.

In this work we investigate potential candidates for arithmetic in Vadalog and show that extending KG reasoning languages with arithmetic is a non-trivial problem. We define a new language as extension of Vadalog's core language with a well-behaving and efficient form of arithmetic. We prove P-completeness of our language, which we call *Warded Bound Datalog*$_{\mathbb{Z}}$. Moreover, we show the first expressivity results for a decidable language supporting arithmetic. In particular, we prove capture results for Limit Datalog$_{\mathbb{Z}}$, a language that was recently introduced in a remarkable line of work by Kaminski et al. Our contributions prove that our language is a suitable candidate for reasoning in AI systems. Thereby, we lay the theoretical foundation for highly expressive logical languages that combine the power of complex recursive reasoning and arithmetic, while maintaining efficient reasoning algorithms for applications in modern AI systems, such as KGs.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Contents

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

CHAPTER 1

Introduction

The resurgence of declarative programming and specifically Datalog [DEGV01] has been extensively observed in research recently. Especially for systems supporting data analytic tasks, ranging from aggregation of data to complex query answering, Datalog is a key aspect [ACC+10, CMA+12, Mar14, SGL15, SYI+16, WBH15]. Some of the reasons for this growing interest in Datalog are its scalability and that it has been argued that the declarative programming paradigm allows users to express complex queries in a more intuitive way than the imperative paradigm. Academics have repeatedly advocated that programs in data analysis should be expressed with declarative programming (Datalog) [ACC+10, SYI+16] and that systems should support this paradigm in their language. Moreover, Datalog supports full recursion, which is needed to express common and complex problems in Big Data analytics. Consequently, Datalog is a prime candidate for a broad variety of research areas such as parallel and distributed systems [Hel10], Big Data systems [ABC+11], and ontological query answering in the area of the Semantic Web [GOP11] amongst others. Ordinary Datalog, however, is not powerful enough to express use cases in such complex scenarios. Thus, a major field of current research is to find languages that extend Datalog with properties to make it more powerful, but at the same time carefully restrict the syntax to keep the complexity of reasoning provably low. In this work we tackle the problem of identifying a new Datalog language that supports arithmetic and can be used for reasoning in Knowledge Graphs. We prove the complexity of our language, present novel reasoning algorithms and compare our language with existing Datalog languages.

Reasoning in KGs. The context of this work is mainly set in the realm of Knowledge Graphs and reasoning in such AI systems. Knowledge Graphs (KGs) represent knowledge in graph-based models and allow for reasoning over the knowledge base with the use of mathematical methods. KGs support traditional logic-based reasoning as well as machine learning (ML) and statistical reasoning and thus enable to manage and leverage

knowledge in a versatile and powerful way. There is no standard definition of KGs in academia. The term Knowledge Graph was coined by Google's KG [Sin12] and may refer to triple-based models or to other structures representing entities and relations. In the following we will use KG mainly when talking about the Vadalog system, which is a KG system supporting relations of arbitrary arity, generalizing triple-based scenarios [BSG18]. Bellomarini, Sallinger and Gottlob (BSG) have identified three minimum requirements a reasoning language for KGs needs to fulfil [BSG18]:

(i) Full recursion over KGs,
(ii) ontological reasoning over KGs, and additionally
(iii) low complexity of reasoning.

Requirement (i) states that the language needs to be able to support full recursion and joins. These features are needed for navigating and reasoning over graph structures. Ordinary Datalog fulfils this requirement, hence the requirement could be equally stated as the language should (at least) encompass Datalog. Ontological query answering (ii) is needed to express common data analytic tasks and the exact requirement is that the language must be able to express SPARQL queries under the OWL 2 QL entailment regime [GOH+13]. Requirement (iii) can be further specified by requiring reasoning to be tractable in data complexity. Hence, algorithms computing fundamental reasoning problems in polynomial time or even more efficiently (e.g., in NLogSPACE) are needed.

Datalog for Scalable Analytics. There are several variations or *fragments* of the Datalog language, which extend ordinary Datalog with features (syntactically or semantically), in order to fulfil the aforementioned requirements. There is a long line of work investigating various fragments and their complexity [BLMS11, CGK13, CGL09b, CGL+10]. BSG, however, observed that most of these fragments do not fulfil all the criteria and are therefore unsuitable for reasoning over KGs. This is why the Vadalog language, based on the Warded Datalog$^{\pm}$ fragment, was introduced. Warded Datalog$^{\pm}$ extends Datalog with features that enable complex reasoning (existential quantification in rule heads) but at the same time restricts the syntax in order to keep the complexity of the language low [BGPS19]. While the Vadalog language can handle reasoning with existential rule heads efficiently and has strong theoretical underpinnings (i.e. polynomial data complexity), there is a vast number of additional features needed, such as arithmetic and aggregation, equality in rule heads, and access to external functions (e.g., ML) [GPS19]. A naive combination of these advanced features with the core language immediately leads to undecidability of the language (even the combination of only one additional feature and the core language leads to undecidability). Undecidability implies inefficient program evaluation and bad worst-case behaviour of reasoning algorithms in general. Practical data-intensive applications and reasoning systems that implement undecidable languages would potentially suffer from bad computational behaviour and in the worst case from non-terminating algorithms. Therefore, complexity theoretic results that pin down the complexity of Datalog fragments are vital in theory and practice. In order to achieve this goal, Datalog fragments and careful restrictions of these fragments that strike the

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

perfect balance between the expressive power of the language and efficiency need to be identified. This is clearly a highly non-trivial undertaking and several approaches have been proposed, but all of them fail to make the language powerful enough to support both reasoning over KGs and arithmetic. Furthermore, most existing Datalog fragments that are capable of arithmetic and aggregation are either undecidable or introduce strict monotonicity requirements and are hence very restrictive. Consequently, it is vital to identify fragments that are powerful enough to extend the query language in a reasonable way and at the same time are also carefully restricted, to allow for good computational complexity results.

Arithmetic in Datalog. Arithmetic and aggregation in recursion play a crucial role for elementary data analytic problems e.g., for computing shortest paths in a graph or bill of materials queries. There is a long line of work that investigates extensions of Datalog with arithmetic and aggregation dating back to the late 1990's [BNST91, CM90, GGZ95, Gel92, KS91, MPR90, RS92] that is experiencing renewed academic interest [FPL11, MSZ13b, ZYD+17]. Earlier approaches mainly try to establish semantics that bring together recursion and arithmetic in Datalog. These semantic mechanisms, however, wrestle with the problem of monotonicity. The semantics of Datalog is based on monotonicity (with respect to set containment) properties of its least fixpoint operator, while arithmetic and aggregates in general violate these monotonicity requirements. This leads to a twofold problem of such approaches: Either they rely on strong monotonicity requirements, which restrict the arithmetic and aggregate functions, or they are undecidable. These problems (most notably undecidability) also carry over to systems implementing the aforementioned theoretical solutions. Prominent examples are DeALS [SYZ15], BOOM [ACC+10], LogicBlox [AtCG+15], and SociaLite [SGL15]. With the goal of tackling these problems, Kaminski et al. proposed limit Datalog$_\mathbb{Z}$, which is a fragment of Datalog with integer arithmetic. Since Datalog extended with integer arithmetic was shown to be undecidable in general [DEGV01], Kaminski et al. introduce restrictions in order to obtain a decidable fragment of Datalog$_\mathbb{Z}$. They show that by opposing further restrictions on their language, a tractable (polynomial time) fragment can be obtained. On the downside, limit Datalog$_\mathbb{Z}$ is - as all other arithmetic approaches proposed so far - not powerful enough for reasoning in KGs. In particular, it does not fulfil requirement (ii) mentioned above since it does not support existential rule heads. Moreover, limit Datalog$_\mathbb{Z}$ does not allow aggregates such as sum and count and its semantics is expressed in an implicit way in programs, rendering limit programs not as well readable as programs with an explicit syntax.

1.1 Problem Formulation

From the notions introduced above it is evident that there is a gap in current research between Datalog languages that support ontological reasoning with existential rule heads and are therefore used for reasoning over KGs (specifically Vadalog) and fragments of Datalog$_\mathbb{Z}$ that support arithmetic but do not support advanced reasoning over KGs. To the best of our knowledge, there are no formal results that unify these two lines of work

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

and hence lay the theoretical foundations for complex data analytic tasks over KGs. More generally, there are no results that prove the complexity of a Datalog language supporting both existentials in the rule heads and some form of arithmetic. In order to provide sufficiently efficient algorithms, which enable well-behaving practical implementations, theoretical results for suitable reasoning languages are needed. More specifically, it is of paramount importance to establish complexity bounds on fundamental reasoning problems of Datalog fragments in order to pin down the complexity of the languages. Such results have a major impact on practical implementations, since naively implementing language fragments with no theoretical underpinnings may lead to very inefficient systems. Formal results such as hardness of language fragments or upper bounds are essential to guarantee the existence of efficient algorithms or to identify fragments for which no such algorithm can exist.

1.2 Research Questions

In this work, we close the gap between KG reasoning languages and arithmetic in data analytics. We use Vadalog and its core language as an exemplary representative for KGs and reasoning languages. Our contributions lay the foundations for powerful Datalog fragments that can be used for efficient reasoning incorporating arithmetic. Naturally, a class of such languages has a broad range of applications in data analytics and logic programming. Moreover, we prove novel results in descriptive complexity theory that relate modern formalisms to deep theoretic results. In more detail, apart from discussing current state-of-the-art reasoning languages for data analytic tasks and KGs, we tackle the following main questions that motivate us. Since there are no complexity results for arithmetic in Vadalog - or any KG reasoning language - we firstly want to investigate simple, straight forward combinations of various language fragments in the context of Vadalog and reasoning in KGs. Because of the novelty of Vadalog, there are no such results in state-of-the-art research.

Question 1. Are naive extensions of Vadalog with arithmetic decidable?

There is a clear gap in current research between Datalog languages that support arithmetic and fragments used for reasoning over KGs. Therefore, we want to develop a new fragment that closes this gap. Our goal is to define a $\text{Datalog}_{\mathbb{Z}}$ language that is powerful enough to be used for reasoning over KGs. Specifically, we strive to define the first fragment that fulfils all the requirements for KG reasoning discussed above, and that additionally supports arithmetic with the potential of being extended to common aggregate functions. It is of utter importance that such a language is decidable for both theory and potential implementations. There is no such language yet, and generally speaking, no complexity result for any combination of existential in rule heads and arithmetic in Datalog is known until today.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

Question 2. Is it possible to define a decidable Datalog language that is capable of both reasoning over KGs and arithmetic?

Even though there has been a lot of research on the complexity of the Vadalog language and several variants of the core language, there are no results of the language extended with arithmetic (as remarked in [GPS19]). More generally, the complexity of arithmetic and aggregation in KG reasoning languages is still wide open. As discussed above, many state-of-the-art reasoning systems rely on formalisms with weak or almost no theoretical guarantees. Our aim is to prove the first complexity results for reasoning with a language supporting arithmetic and reasoning over KGs.

Question 3. What is the complexity of a decidable reasoning language for KGs (Vadalog) that supports arithmetic?

The results of Kaminski et al. on the complexity of arithmetic in $\text{Datalog}_{\mathbb{Z}}$ do not explicitly prove the expressive power or connections to descriptive complexity theory in general. This, however, would greatly improve our understanding of the interplay between logic, arithmetic, and computational complexity. As a result, it is essential to prove connections between limit $\text{Datalog}_{\mathbb{Z}}$ and descriptive complexity. Thereby we could also answer a question about the expressive power of limit $\text{Datalog}_{\mathbb{Z}}$ mentioned by recently by Grau et al. in [GHK+19].

Question 4. What is the expressive power of a Datalog language with decidable arithmetic? In particular, what is the expressive power of limit $\text{Datalog}_{\mathbb{Z}}$?

1.3 Results

Firstly, we show that naive combinations of the Vadalog language, and several relevant fragments thereof are undecidable. In particular, we investigate Warded $\text{Datalog}^{\pm}$ extended with arithmetic, piece-wise linear Warded $\text{Datalog}^{\pm}$ extended with arithmetic and piece-wise linear Warded $\text{Datalog}^{\pm}$ extended with limit arithmetic. We show that all these fragments are undecidable. On an intuitive level, these results strongly emphasize that it is non-trivial to find a language that is powerful enough for reasoning in KGs and supports arithmetic. Furthermore, these negative results motivate us to find a suitable Datalog fragment with decidable reasoning.

Theorem 1.3.1. (Informal Statement). A naive extension of Warded $\text{Datalog}^{\pm}$ with arithmetic is undecidable.

As a first step towards a new reasoning language, we firstly review the limit arithmetic introduced by Kaminski et al. and discuss why their proposed semantics are not ideal e.g., for users. Note that their fragment is to the best of our knowledge the only decidable Datalog language supporting arithmetic where also a tractable fragment has

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

been identified. We define a syntactic fragment using some of their ideas and prove that we can leverage their interesting formal techniques in order to obtain complexity guarantees for our syntactic fragment. We show decidability and coNP-completeness of our language, called *bound Datalog*$_{\mathbb{Z}}$.

Theorem 1.3.2. (Informal Statement). Bound Datalog$_{\mathbb{Z}}$ is coNP-complete.

Since we somewhat build on ideas proposed in the context of limit Datalog$_{\mathbb{Z}}$, we also investigate the expressive power of positive limit Datalog$_{\mathbb{Z}}$. That is, we formally examine the connection between computational complexity and the power of this logic language. Such investigations generally fall into the area of *descriptive complexity theory*. One of our central contributions is that we prove expressive power results for limit Datalog$_{\mathbb{Z}}$, which show that limit Datalog$_{\mathbb{Z}}$ captures coNP, meaning that every property decidable by a coNP algorithm can be expressed as a limit Datalog$_{\mathbb{Z}}$ program. This provides deep insight into arithmetic in logic and logic programming, and its connection to computational complexity. To the best of our knowledge, this is the first expressive power result for a decidable Datalog$_{\mathbb{Z}}$ language capable of arithmetic.

Theorem 1.3.3. (Informal Statement). Limit Datalog$_{\mathbb{Z}}$ captures coNP.

In order to bridge the gap between arithmetic and reasoning in KGs, which is the central goal of this thesis, we design a carefully restricted extension of the core Vadalog language with our bound fragment which we call *Warded Bound Datalog*$_{\mathbb{Z}}$. To the best of our knowledge this is the first well-defined notion of a language supporting arithmetic and being powerful enough for reasoning in KGs. Hence, we define the first (Datalog-based) reasoning language consisting of a combination of Datalog extended with both arithmetic and existential rule heads. Finally, we present an algorithm for reasoning in Warded Bound Datalog$_{\mathbb{Z}}$ based on a reasoning algorithm with a termination strategy for Warded Datalog$^{\pm}$ and ideas from our bounded fragment. This result does not only provide us with respective complexity bounds on reasoning with our new language but can also be seen as a reasonable basis for a potential practical implementation in KG systems such as Vadalog.

Theorem 1.3.4. (Informal Statement). Warded Bound Datalog$_{\mathbb{Z}}$ is P-complete.

Note that Theorem 1.3.1 refutes Question 1, Theorem 1.3.4 provides answers to Question 2 and Question 3, and finally, Theorem 1.3.3 answers Question 4.

Summary of Main Results. Our results encompass the following main aspects:

- We prove several undecidability results that show limitations of naive combinations of arithmetic and complex reasoning in KGs
- We propose a syntactic fragment of arithmetic in Datalog (denoted *bound Datalog*$_{\mathbb{Z}}$), based on the ideas of Kaminski et al. and their limit Datalog$_{\mathbb{Z}}$ fragment. We show that we can leverage techniques of Kaminski et al. for this purely syntactic fragment in order to obtain complexity results

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

- We show new descriptive complexity results that prove the expressive power of limit arithmetic i.e., we show that limit Datalog$_{\mathbb{Z}}$ captures coNP
- We prove the first complexity result for programs with arithmetic and existential heads in Datalog. We not only show decidability but even tractability of our language Warded Bound Datalog$_{\mathbb{Z}}$
- We give an efficient reasoning algorithm with practical potential for our new KG reasoning language, Warded Bound Datalog$_{\mathbb{Z}}$, which supports arithmetic

1.4 Organization

The rest of this work is organized as follows. We begin the technical part of this thesis with some preliminary definitions and an overview over techniques we use in Chapter 2. In Chapter 3 we give an outline of current research areas and important results such as several Datalog fragments relevant for the rest of this work. The heart of our technical work is Chapter 4, where we prove our main theorems. Section 4.1 covers negative results that discuss boundaries of naive approaches. In Section 4.2 we introduce our syntactic fragment, in Section 4.3 we prove descriptive complexity results of limit Datalog$_{\mathbb{Z}}$ and in Section 4.4 we show how one can integrate arithmetic in Vadalog. Finally, we conclude with a brief discussion and comparison in Section 4.5 and give a summary and an outlook on possible future research in Chapter 5.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

CHAPTER 2

Preliminaries

In this section we give fundamental definitions needed throughout the rest of this work. We assume basic knowledge of logic and model theory. In particular, we assume familiarity with first-order logic, logic programming and finite model theory. Furthermore, we assume that the reader is familiar with basic complexity theoretic aspects, for instance complexity classes and reductions. This chapter is organized as follows. In Section 2.2 we give complexity theoretic definitions of Turing machines and complexity classes. In Section 2.3 we introduce important notions from Datalog and logic programming in general. These aspects are extended in Section 2.4 to a database-theoretic context. Finally, in Section 2.5 and Section 2.6 we discuss essential aspects around Vadalog and arithmetic in Datalog, which lay the foundation of our contributions.

2.1 Notation

We write tuples $(t_1, \ldots, t_n)$ as $\mathbf{t}$. By slight abuse of notation, we sometimes write $\mathbf{t}$ to indicate the set $\{t_i \mid t_i \in \mathbf{t}\}$ and use e.g., $t_i \in \mathbf{t}$ and $|\mathbf{t}|$. We use $\models$ to denote semantic consequence (i.e., logical entailment). When we do not specify an encoding we refer to some canonical binary encoding of an object and if x is some object (e.g., an integer) $\llcorner x \lrcorner$ denotes the binary encoding of x, i.e., some canonical encoding over the language $\{0,1\}^*$. With $||\llcorner x \lrcorner||$ we denote the size of a binary encoding of S and with $|S|$ we denote the number of elements in S. Sometimes we also drop the $\llcorner\lrcorner$ and denote with x both the object and its representative (binary) encoding. We write complexity classes as CLASS, names for algorithmic problems as Problem, and names for important relations or sets as importantset. We use syntactic shorthand notation $R(x) \doteq 0$ for $R(x) \leq 0 \wedge R(x) \geq 0$, $R(x \geq y)$ for $R(y \leq x)$ and $R(x \leq y \leq z)$ for $R(x \leq y) \wedge R(y \leq z)$.

2.2 Computational Complexity

In this section we present definitions of Turing Machines, basic complexity classes, and reductions that are needed throughout this work, we thereby follow [AB09].

2.2.1 Turing Machines

In general, we use Turing Machines (TMs) to define computational properties of complexity classes. Some of the most fundamental ones are the class of problems solvable in polynomial time, P, and the class of problems solvable in non-deterministic polynomial time, NP. Note that for the definitions we talk about languages and TMs accepting them, as problems are usually encoded as languages over $\{0,1\}^*$ and the question asked is if a (encoding of a) problem instance is in the respective language or not. In the rest of this work, we assume that all mentioned functions are time-constructible.

A k-tape Turing machine $M = (\Gamma, Q, \delta)$ is defined as follows.

- Γ is a set of symbols that the tapes of M can contain. We assume that Γ contains a special blank symbol, $\square$, a designated start symbol, $\triangleright$, and numbers 0 and 1;
- The set Q holds the states M's registers can be in. We assume that $q_{start}, q_{halt} \in Q$;
- The transition function $\delta : Q \times \Gamma^k \mapsto Q \times \Gamma^k \times \{L, S, R\}^k$, which describes the rules M uses to perform stepwise computation.

If M is in state q and the tuple of symbols currently on the k tapes is $(\sigma_1, \ldots, \sigma_k)$ and $\delta(q, (\sigma_1, \ldots, \sigma_k)) = (q', (\sigma'_1, \ldots, \sigma'_k), z)$ for a $z \in \{L, S, R\}^k$, then symbols σ will be replaced by symbols σ' at the next step, M will be in state q' and the k heads will move left, right or stay at their position, as given by z. Initially all tapes except for one, which contains the input, have the start symbol in the first location and the blank symbol in all other locations. The tape containing the input has the start symbol in its first location followed by the finite input string (not containing blank symbols) and the rest of tape contains blank symbols. The start configuration of M is defined by all heads of M being on the left most position of the tapes and M is in state q_{start}. The state q_{halt} is the halting state of M, specified by the property that δ disallows any further changes of states or modifications of the tapes. Let us now define the measure of complexity of our model of computation. Intuitively, the complexity of a TM M computing a function is given by the number of steps M performs in order to compute its output.

Definition 2.2.1. Let $f : \{0,1\}^* \mapsto \{0,1\}^*$ and $T : \mathbb{N} \mapsto \mathbb{N}$, and let M be a Turing machine. We say M computes f in time T-time if $\forall x \in \{0,1\}^*$, M halts after at most $T(|x|)$ steps with $f(x)$ written on its output tape.
Turing machine M computes f if it computes f in T-time for some $T : \mathbb{N} \mapsto \mathbb{N}$.

2.2.2 Complexity Classes

We are now ready to formally define complexity classes, which are sets of languages that can be computed with a certain (asymptotic) time bound.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

Definition 2.2.2. Let $T : \mathbb{N} \mapsto \mathbb{N}$ be some function. The set of all Boolean functions that are computable in $c \cdot T$-time for some constant $c > 0$ is denoted $\mathsf{DTIME}(T(n))$.
The class of polynomial-time decision problems is defined as

$$\mathsf{P} = \bigcup_{c \geq 1} \mathsf{DTIME}(n^c)$$

DTIME refers to deterministic time, hence these languages can be computed by deterministic TMs. The class P is generally associated with efficiently computable languages. Informally, NP is the class of languages whose correctness is easy to verify. Hence, provided with a possible solution to a problem instance, a certificate, it is computably easy to verify if the certificate is a correct solution of the problem.

Definition 2.2.3. A language $L \subseteq \{0,1\}^*$ is in NP if there exists a polynomial $p : \mathbb{N} \mapsto \mathbb{N}$ and a polynomial time TM M s.t. for every $x \in \{0,1\}^*$,

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

Note that as it is allowed that $p(|x|) = 0$ (u is empty) it follows directly that $\mathsf{P} \subseteq \mathsf{NP}$ by definition. Non-deterministic computation is characterized by a variant of ordinary Turing machines called non-deterministic Turing machines (NDTM). These TMs have two transition functions and an additional special state, q_{accept}. A non-deterministic TM is said to accept the input if the accepting state q_{accept} is reached by some computation branch of the machine. When an NDTM M computes a function, it makes an arbitrary decision at each step, deciding which transition function to apply. Machine M accepts an input if there exists a sequence of such choices that makes M reach an accepting state. Conversely, M rejects if all such sequences of choices reach a halting state without reaching an accepting state. We say M runs in time T if for every input $x \in \{0,1\}^*$ and every sequence of non-deterministic choices, M reaches a halting state or an accepting state within $T(|x|)$ steps.

Let the complement of a language $L \subseteq \{0,1\}^*$ be denoted as $\bar{L}$, i.e., $\bar{L} = \{0,1\}^* \setminus L$. Intuitively, NP computation can be thought of as certifying computation, in the sense that the accepting computation path of a NP TM serves as certificate for an element of the language it decides. The following definition provides us with the complexity class that defines languages whose complements can be certified in NP.

Definition 2.2.4. $\mathsf{coNP} = \{L : \bar{L} \in \mathsf{NP}\}$

It is essential to say that coNP is not the complement of NP. In fact, $\mathsf{NP} \cap \mathsf{coNP} \neq \emptyset$ and P is contained in this intersection. A wide-open question in complexity is if it holds that $\mathsf{NP} = \mathsf{coNP}$. It is widely believed that these classes are in fact unequal. An alternative definition for coNP in a similar fashion as the definition of NP would be to define coNP as the class of languages for which a polynomial time TM M accepts an input for *all* certificates. The exponential time analogues for P and NP are defined in a similar fashion as their polynomial equivalents, i.e., $\mathsf{EXP} = \bigcup_{c \geq 0} \mathsf{DTIME}(2^{n^c})$ and $\mathsf{NEXP} = \bigcup_{c \geq 0} \mathsf{NTIME}(2^{n^c})$.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

2.2.3 Reductions

In order to prove hardness results of problems, it is generally useful to use the notion of reductions. Thus, for instance by (polynomial-time Karp-) reducing an NP-hard problem to a new problem, we can conclude that the new problem is also NP-hard. We give an exemplary definition of polynomial-time Karp reductions.

Definition 2.2.5. A language $\mathcal{L} \subseteq \{0,1\}^*$ is polynomial-time Karp reducible to $\mathcal{L}' \subseteq \{0,1\}^*$, denoted as $\mathcal{L} \leq_m^p \mathcal{L}'$, if there is a polynomial-time computable function $f : \{0,1\}^* \mapsto \{0,1\}^*$ s.t. for every $x \in \{0,1\}^*, x \in \mathcal{L} \Leftrightarrow f(x) \in \mathcal{L}'$.

We can also generalize this notion of reducability to functions of various complexity classes. We write $\mathcal{L} \leq_m^C \mathcal{L}'$ if there is a function $f \in C$ s.t. $x \in \mathcal{L} \Leftrightarrow f(x) \in \mathcal{L}'$, where C is a complexity class. There is a more powerful notion of reducability that uses oracle TMs to prove a result. Informally, oracle Turing Machines are TMs that have access to an oracle that can act as black box for a certain problem and can answer questions about the respective language in one computational step. For instance, a TM can ask an oracle for a language $\mathcal{L}$: "is x in $\mathcal{L}$" and the oracle will reply with the correct answer in one step. Results obtained in such a way are called *relativized*. In general, the language $O \subseteq \{0,1\}^*$ is called the oracle for a TM. We say that a language $\mathcal{L}$ is Turing reducible to $\mathcal{L}' \subseteq \{0,1\}^*$, denoted as $\mathcal{L} \leq_T^C \mathcal{L}'$, if there is an oracle machine $M \in C$ accepting $\mathcal{L}$ when given oracle $\mathcal{L}'$. Alternatively we write $\mathcal{L} \in C^{\mathcal{L}'}$.

2.3 Datalog Foundations

Let us introduce some basic notions in the area of logic programming. We follow [Llo87]. The theory of logic programs lays the foundation of Datalog languages. We consider countably infinite, disjoint sets of predicates, variables, and constants. The union of the set of variables and the set of constants is the set of *terms*. Each *predicate* P has non-negative arity n, written as P/n for some $n \geq 0$. The position $P[i]$ of an n-ary predicate P for some $i \in [1,n]$ denotes the i-th argument of P. An *atom* is of the form $P(t_1,\ldots,t_n)$ where P is an n-ary predicate and $t_1,\ldots,t_n$ are terms. We write $\mathsf{vars}(\alpha)$ to denote the set of variables occurring in the atom α. Variable free terms are called *ground*. A *rule* r is a FO sentence of the form $\forall \mathbf{x} \phi \rightarrow \psi$ where $\mathbf{x}$ contains all variables in ϕ and ψ and ϕ is called *body* and ψ is called the *head* of the rule. The formula ϕ is a conjunction of atoms and the head consists of one atom. We use $\mathsf{head}(r)$ to denote the atom in the head of r and $\mathsf{body}(r)$ to denote set of body atoms of r. A *fact* is a rule r' with $\mathsf{body}(r') = \emptyset$ and $\mathsf{head}(r')$ contains only constants (similar to a unit clause in classical logic programming [Llo87]). A logic *program* $\mathcal{P}$ consists of a set of rules whose variables are assumed to be universally quantified (we usually do not write the universal quantifiers explicitly). A *substitution* is a function $h : T \mapsto T'$ where T, T' are sets of terms. We write $h_{|S}$ to denote the restriction of h to some $S \subseteq T$: $\{t \mapsto h(t) \mid t \in S\}$.

TU Bibliothek Your knowledge hub WIEN
Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

2.3.1 Semantics of Logic Programs

The semantics of logic programs is defined via traditional notions stemming from mathematical logic such as *interpretations* and *models*. Let us introduce some standard definitions.

Definition 2.3.1. (Herbrand Universe, Herbrand Base) Let $\mathcal{L}$ be a FO language. The *Herbrand universe* is the set $U_{\mathcal{L}}$ of all ground terms constructed from constants and function symbols in $\mathcal{L}$. The *Herbrand base* of $\mathcal{L}$ is the set $\mathcal{HB}_{\mathcal{L}}$ of all ground atoms that can be formed using predicate symbols in $\mathcal{L}$ with ground terms from $\mathcal{U}_{\mathcal{L}}$ as arguments.

The Herbrand base is the set of atoms that can be constructed from symbols of the language with ground terms of the universe. This is used as fundamental notion for interpretations and models of languages that are used to describe the semantics of logic programs, similarly to interpretations in model theory.

Definition 2.3.2. (Herbrand Interpretation) Let $\mathcal{L}$ be a first-order language. The Herbrand interpretation is an interpretation whose domain is $\mathcal{U}_{\mathcal{L}}$, mapping constants $c \in \mathcal{L}$ to c.

Definition 2.3.3. (Herbrand Model) Let $\mathcal{L}$ be a first-order language and S a set of closed formulas of $\mathcal{L}$. A *Herbrand model* for S is a Herbrand interpretation for $\mathcal{L}$ which is a model of S.

For a set S of formulas, we will refer to an interpretation of S rather than to the underlying FO language, assuming that the language is defined by constants and predicate symbols from S. Thus, we refer to Herbrand interpretations of S as subsets of $\mathcal{HB}_S$. Mostly, S will be a program $\mathcal{P}$ and we refer to $\mathcal{U}_{\mathcal{P}}$ and $\mathcal{HB}_{\mathcal{P}}$.

The following definition introduces a forward chaining operator which is applied rules in an iterative way. The least fixpoint of this operator can be used to define the *least fixpoint semantics* of logic programs, which is a notion heavily used in logic programming and Datalog.

Definition 2.3.4. (Immediate Consequence Operator) Let $ground(\mathcal{P})$ be the set of all ground instances of rules in $\mathcal{P}$. The *immediate consequence operator* $T_{\mathcal{P}}$ is a function on Herbrand interpretations of $\mathcal{P}$:

$$T_{\mathcal{P}}(I) = \{\alpha \mid \alpha \leftarrow \beta \in ground(\mathcal{P}) \wedge \{\beta\} \subseteq I\}$$

2.4 Database-theoretic Notions

Let us introduce fundamental definitions specifically related to a database theoretic context. Logic programming and database theory are deeply interconnected. For definitions we follow [BGPS19] in this section. We need the following notions: a *schema* S is a finite set of predicates. An instance J over S is a set of atoms that contains

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

constants and labelled nulls that are equivalent to new Skolem constants, not appearing in the set of terms yet. A *database* over S is a finite set of facts over S. Since queries can be seen as functions, mappings between sets of atoms are important in a database context. An *atom homomorphism* between two sets of atoms A, B is a substitution $h : A \mapsto B$ s.t. h is the identity for constants and $P(t_1, \ldots, t_n) \in A$ implies $h(P(t_1, \ldots, t_n)) = P(h(t_1), \ldots, h(t_n)) \in B$. We consider conjunctive queries, which are equivalent to simple SELECT, FROM, WHERE queries in SQL and are of main interest in database theory. A conjunctive query (CQ) over a schema S is a rule of the form

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

where the predicate Q is used in the head only, $P_1, \ldots, P_n$ are atoms without nulls and $\mathbf{x} \subseteq \bigcup z_i$ are the *output variables*. A Boolean conjunctive query (BCQ) is a CQ where the head predicate Q is of arity 0, hence contains no (free) variables in $\mathbf{x}$. The answer to a BCQ q is *yes* if the empty tuple () is an answer of q.

The *evaluation* of a CQ $q(\mathbf{x})$ over an instance J is the set of all tuples of constants induced by a homomophism $h(\mathbf{x})$, where $h : \mathsf{atoms}(q) \mapsto J$. We mostly consider an extension of traditional logic programs, which allow existential quantification in rule heads. These types of rules are called *Tuple Generating Dependencies* in database theory.

Definition 2.4.1. (Tuple Generating Dependencies, TGDs) A Tuple Generating Dependency σ is a first order sentence of the form

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

where $\mathbf{x}$,$\mathbf{y}$,$\mathbf{z}$ are tuples of variables and ϕ, ψ are conjunctions of atoms without constants and nulls. Equivalently to standard Datalog rules, $\mathsf{body}(\sigma)$ and $\mathsf{head}(\sigma)$ denote the body and head of the rule. The frontier of σ is defined as $\mathsf{frontier}(\sigma) = \mathsf{vars}(\phi) \cap \mathsf{vars}(\psi)$. The set of existential variables in σ is $\mathsf{var}_\exists(\sigma) = \mathsf{vars}(\psi) \setminus \mathsf{frontier}(\sigma)$. The set of predicates in Σ is denoted $\mathsf{sch}(\Sigma)$.

For brevity we typically write a TGD $\phi(\mathbf{x}, \mathbf{y}) \rightarrow \exists \mathbf{z} \psi(\mathbf{x}, \mathbf{z})$ and use comma instead of $\wedge$ to indicate conjunctions (and joins respectively). A TGD σ is *applicable* w.r.t. an interpretation I if there exists a homomorphism h s.t. $h(\mathsf{body}(\sigma)) \subseteq I$. If a TGD is applicable to an interpretation, it can be applied, deriving the head atom of the TGD. Having the definition of rule applicability at hand, we can now define the formal, model-theoretic semantics of TGDs w.r.t. an interpretation.

Definition 2.4.2. (Semantics of TGDs) An instance J satisfies a TGD σ, denoted $J \models \sigma$, if the following holds: If σ is applicable w.r.t. J, i.e., there is a homomorphism h in σ s.t. $h(\phi(\mathbf{x}, \mathbf{y})) \subseteq J$ then there is a homomorphism $h' \supseteq h_{|\mathbf{x}}$ s.t. $h'(\psi(\mathbf{x}, \mathbf{z})) \subseteq J$.
An instance J satisfies a set of TGDs Σ, $J \models \Sigma$, if $J \models \sigma$ for each $\sigma \in \Sigma$.

Since we are concerned with the complexity of reasoning with logical languages, we need to define fundamental reasoning tasks whose complexity we use as measure. These are

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

the main problems we use in our complexity theoretic investigations. Conjunctive query answering is one of the fundamental reasoning tasks of TGDs [BGPS19].

Definition 2.4.3. (**CQ answering under TGDs**) Given a database D and a set of TGDs Σ, an instance J is a *model* of D and Σ s.t. $J \supseteq D$ and $J \models \Sigma$. We use $\mathsf{mods}(D, \Sigma)$ to denote the set of all models of D, Σ.

One of the main tasks considered in database systems is computing the *certain answers*, $\mathsf{cert}(q, D, \Sigma)$ to a query q, which is usually defined as the set of tuples occurring in all models of a database and a set of TGDs: $\mathsf{cert}(q, D, \Sigma) = \bigcap_{J \in \mathsf{mods}(D,\Sigma)} q(J)$. For complexity theoretic questions we consider the corresponding decision problem of deciding whether for a tuple $\mathbf{c}$, $\mathbf{c} \in \mathsf{cert}(q, D, \Sigma)$ and denote this decision problem CQAns(Σ).

Chase Procedure. The standard technique that can be applied to solve the problem of computing certain answers is the *chase procedure* [JK82, MMS79]. This is a forward chaining approach to compute the certain answers to a query. We let $\mathsf{chase}(J, \Sigma)$ denote the result of applying the chase procedure for an instance J under a set of TGDs Σ (i.e., a universal model which is homomorphically embeddable into every other model of D and Σ). A standard result in database theory is that given a database D and a CQ q, $\mathsf{cert}(q, D, \Sigma) = q(\mathsf{chase}(D, \Sigma))$.

A central problem in database theory is the problem of checking whether a certain tuple can be derived from a database and a set of rules, i.e., whether the database and the set of rules entail a certain tuple. This problem is usually called query evaluation.

Definition 2.4.4. (**BCQ Evaluation Problem**, BCQEval) Given a BCQ q, a set of TGDs Σ and a database D, the BCQ evaluation problem is to decide whether the empty tuple () is entailed:

$$D \cup \Sigma \models q$$

Note that showing that $D \cup \Sigma \models q \iff$ showing $D \cup \Sigma \cup \{\neg q\}$ is an unsatisfiable theory (i.e., an unsatisfiable set of first-order sentences) [CGK13]. A classical result in database theory shows that two of the most fundamental problems, CQAns and BCQEval are LogSPACE-equivalent [CGK13].

Another fundamental problem that is often used to prove complexity results about Datalog languages is the problem of deciding whether a database D and a set of rules Σ entail a certain fact.

Definition 2.4.5. (**Fact Entailment Problem**, Factent) Given a program $\mathcal{P} = D \cup \Sigma$ and a fact α, the problem of deciding whether $\mathcal{P} \models \alpha$ is denoted Factent.

Note that since we can encode a CQ as a rule (a CQ is essentially a rule), the notion of fact entailment subsumes conjunctive query answering [Kos20, BLMS11]. The following easy to verify proposition formalizes this intuition.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

Proposition 2.4.6. BCQEval $\leq_m^p$ Factent.

Proof. Let $Q = \phi(\mathbf{x}) \rightarrow q$ be a BCQ. We construct an instance of Factent (P, α): $P = \phi(\mathbf{x}) \rightarrow q$ and $\alpha = q$.

- ($\Longrightarrow$) If Q is a positive instance of BCQEval then there exists a homomorphism $h : h(\phi(\mathbf{x})) \in I$ for an interpretation I. Then $I \models P$ implies $I \models \alpha$, thus $P \models \alpha$.
- ($\Longleftarrow$) If $P \models \alpha$ then $I \models \alpha$ whenever $I \models P$. Hence, $I \models \phi(\mathbf{x})$ and $q \in I$.

The reduction is clearly computable in polynomial time. □

As with problems above, we consider both data complexity and combined complexity of Factent:

- Combined complexity: we assume $\mathcal{P}$ and α are part of the input
- Data complexity: we assume $\mathcal{P}$ to be given as a dataset $\mathcal{D}$ and a program $\mathcal{P}'$: $\mathcal{P} = \mathcal{P}' \cup \mathcal{D}$ where only $\mathcal{D}$ and α are part of the input and $\mathcal{P}'$ is fixed.

2.5 Logical Core of Vadalog

The logical core of Vadalog is an extension of Datalog$^{\pm}$ which is a family of Datalog languages with existentials in rule heads and restrictions in order to make the language decidable. This fragment restricts the way nulls can be propagated when applying TGDs. In this section we give the basic definitions of this fragment, called Warded Datalog$^{\pm}$. Moreover, we review the way the Vadalog language restricts recursion leading to the definition of piece-wise linear Warded Datalog$^{\pm}$. We review the most important definitions and theorems around Warded Datalog$^{\pm}$ proposed in [BGPS19]. Warded Datalog$^{\pm}$ restricts the syntax of TGDs in order to control the use of so-called dangerous variables that can be unified null values in existential rule heads during program evaluation. It introduces wards, which are designated body atoms that firstly cover all dangerous variables that might be unified with a null value during program evaluation and secondly, the ward restricts the way the rest of the body can interact with those values. The goal is to restrict derivation of predicates with null values, since it has been shown that unrestricted use of dangerous variables has a direct relation to high complexity of reasoning [CGK13]. In order to define wardedness we need several basic definitions. We define the set of *positions* of a schema S, $\mathsf{pos}(S)$ as the set $\{P[i] \mid P/n \in S, i \in [1, n]\}$. We write $\mathsf{pos}(\Sigma)$ to denote the respective set over a set of TGDs Σ.

Definition 2.5.1. (Affected Positions) The set of *affected positions* of $\mathrm{sch}(\Sigma)$, denoted $\mathsf{affected}(\Sigma)$ is defined inductively:

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

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

The set of non-affected positions is defined as: $\mathsf{nonaffected}(\Sigma) = \mathsf{pos}(\Sigma) \setminus \mathsf{affected}(\Sigma)$.

Depending on the position variables occur in rules and how they "interact" with nulls and other variables, we can classify them into three stages.

Definition 2.5.2. (Harmless, Harmful and Dangerous Body Variables) For a TGD $\sigma \in \Sigma$ and a variable $x \in \mathsf{body}(\sigma)$:

- x is *harmless* if at least one occurrence of it is in $\mathsf{body}(\sigma)$ at a position $\pi \in \mathsf{nonaffected}(\Sigma)$
- x is *harmful* if it is not harmless
- x is *dangerous* if it is harmful and $x \in \mathsf{frontier}(\sigma)$

We are now ready to formally define wardedness.

Definition 2.5.3. (Warded TGDs) A set Σ of TGDs is *warded* if either

1. for each $\sigma \in \Sigma$, there are no dangerous variables in $\mathsf{body}(\sigma)$, or
2. there is an atom $\alpha \in \mathsf{body}(\sigma)$, called *ward*, s.t. (i) all the dangerous variables in $\mathsf{body}(\sigma)$ occur in α and (ii) each variable of $\mathsf{vars}(\alpha) \cap \mathsf{vars}(\mathsf{body}(\sigma) \setminus \{\alpha\})$ is harmless

Let WARD denote the family of all finite warded sets of TGDs. Reasoning with warded TGDs is tractable in data complexity. More formally, CQAns(WARD) is EXP-complete in combined complexity and P-complete in data complexity [AGP14]. Besides limiting the way variables can interact with atoms in TGDs, Vadalog employs another restriction which controls the way recursion is allowed by warded rules. The intuition behind this idea is that each TGD has at most one atom in the body of the rule whose predicate is mutually recursive with a predicate occurring in the head of the TGD. This type of recursion is called *piece-wise linear* recursion and has already been investigated in the context of Datalog. In order to define piece-wise linearity we need to define some preliminary concepts. The *predicate graph* of a set of TGDs Σ, denoted $pg(\Sigma)$ is a directed graph $G = (V, E)$ where $V = \mathsf{sch}(\Sigma)$. The edge relation is defined as follows: $E = \{(P, R) \mid \sigma \in \Sigma : P \in \mathsf{body}(\sigma) \text{ and } R \in \mathsf{head}(\sigma)\}$. Two predicates P, R are *mutually recursive* if there exists a cycle in the predicate graph containing P and R.

Definition 2.5.4. (Piece-wise linear TGDs) A set of TGDs Σ is *piece-wise linear* if for each $\sigma \in \Sigma$ there exists at most one atom in $\mathsf{body}(\sigma)$ whose predicate is mutually recursive with a predicate in $\mathsf{head}(\sigma)$. We let PWL denote the family of set of piece-wise linear TGDs.

By restricting warded TGDs to be piece-wise linear the data complexity of reasoning can be pushed down even further and in [BGPS19] the authors show that CQAns(WARD $\cap$ PWL) is PSPACE-complete in combined complexity and NLogSPACE-complete in data complexity. Moreover, the combination of piece-wise linearity and wardedness is reasonable since reasoning with piece-wise linear rules is undecidable in general [BGPS19].

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

2.6 Limit Datalog$_\mathbb{Z}$

We review the main definitions and results presented by Kaminski, Grau, Kostylev, Motik and Horrocks (KGKMH) in [KGK+17]. In Datalog$_\mathbb{Z}$ we assume countably infinite (mutually disjoint) sets of objects, object variables, numeric variables, and predicates. Each predicate position is either an object or a numeric position. The set of predicates also contains $\{\leq, <\}$ which we denote comparison predicates. Variables can be *object variables* or *integer variables*. We consider integers in $\mathbb{Z}$ and $\{+, -, \times\}$ are the standard arithmetic functions of addition, subtraction, and multiplication over the standard field of integers, $\mathbb{Z}$. An object term is either an object variable or an object. A *constant* is an object or an integer from $\mathbb{Z}$. A *numeric term* is an integer, a numeric variable or an arithmetic term of the form $t_1 \circ t_2$ where $\circ \in \{+, -, \times\}$ and t_1, t_2 are numeric terms. We distinguish between standard and comparison atoms. Standard atoms are of the form $A(t_1, \ldots, t_n)$ where A/n is a standard predicate and each t_i is a term. Comparison atoms have the form $t_1 \circ t_2$, where t_1, t_2 are numeric terms and $\circ \in \{\leq, <\}$ is a comparison predicate. As in Datalog, a rule r is a sentence $\forall \mathbf{x} \phi \rightarrow \psi$. The body of r, ϕ, is a conjunction of atoms, the head of r is a standard atom and $\mathbf{x}$ denotes the tuple containing all variables in head and body of r. We use $\mathsf{sb}(r)$ to denote the standard body of r, consisting of a conjunction of all standard atoms in the body of r and analogously we use $\mathsf{cb}(r)$ to denote the conjunction of all comparison atoms in the body of r. Equivalently to above we use $\mathsf{head}(r)$ and $\mathsf{body}(r)$ to denote the head and body of r. A fact is a rule that has an empty body and there are no variables or arithmetic functions in the head of the rule.

We adapt the usual Datalog notions to Datalog$_\mathbb{Z}$. An interpretation $\mathcal{I}$ is a set of facts. $\mathcal{I}$ satisfies a ground atom α, $\mathcal{I} \models \alpha$ if one of the following holds:

- α is a standard atom and $\mathcal{I}$ contains each fact obtained from α by evaluating the numeric terms in α;
- α is a comparison atom, evaluating to true under the usual semantics of comparisons

This notion can be extended to conjunctions of ground atoms, rules and programs as in first-order logic. A model is an interpretation that satisfies all rules of a program: $\mathcal{I} \models \mathcal{P}$. We say $\mathcal{P}$ entails a fact, $\mathcal{P} \models \alpha$, if whenever $\mathcal{I} \models \mathcal{P}$ then $\mathcal{I} \models \alpha$ holds. We are specifically interested in fact entailment as a fundamental problem of Datalog languages.

A first result shows that even when only using addition, Factent is still undecidable for Datalog$_\mathbb{Z}$ programs, even if the program contains no multiplication or subtraction and each atom has at most one numeric term. As Factent is undecidable for Datalog$_\mathbb{Z}$ [DEGV01] and is even undecidable if a program contains no multiplication or subtraction and each standard atom has at most one numeric term, Kamninski et al. introduced limit programs. First and foremost, this fragment is a semantic restriction of how numeric values are handled, however as the authors point out, one can axiomatize their limit notion in Datalog$_\mathbb{Z}$. This axiomatization, however, requires constructing atoms for all integers.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

Definition 2.6.1. (Limit Datalog$_\mathbb{Z}$) Predicates in limit Datalog$_\mathbb{Z}$ are either object predicates without numeric positions or numeric predicates where only the last position is allowed to be numeric. Limit Datalog$_\mathbb{Z}$ distinguishes between limit numeric predicates and ordinary (exact) numeric predicates. Limit predicates are either min or max predicates. A Datalog$_\mathbb{Z}$ rule r is a limit rule if either

- $\mathsf{body}(r) = \emptyset$, or
- each atom in $\mathsf{sb}(r)$ is an object, ordinary numeric or a limit atom and $\mathsf{head}(r)$ is an object or limit atom and all exact predicates are EDB

A limit Datalog$_\mathbb{Z}$ program $\mathcal{P}$ is a program that contains only limit rules.

For a limit atom C we write $\preceq_C$ for $\leq$ if C is max and $\geq$ if C is a min limit predicate (with $\prec_C, \succeq_C, \succ_C$ defined accordingly). Essentially, limit predicates are used to keep track of the minimum/maximum integer value for a tuple of a predicate. Let us now formally recall the most important definitions of Kaminski et al. and their limit fragment as they are needed for parts of our work. First, the notion of interpretations must be adapted to capture the semantics of limit predicates. We want to be able to formally express that min/max predicates imply the validity of numeric predicates that are subsumed by the limit notion.

An interpretation $\mathcal{I}$ is *limit-closed* if $C(\mathbf{a}, k') \in \mathcal{I}$ for each limit fact $C(\mathbf{a}, k) \in \mathcal{I}$ and each integer k' s.t. $k' \preceq_C k$. Note that if a limit-closed interpretation contains $B(\mathbf{t}, k)$ then it holds that either

- $B(\mathbf{a}, l)$ for an integer $l \succeq_C k$ and $B(\mathbf{t}, k') \notin \mathcal{I}$ for all $k' \succ_C l$, or
- $B(\mathbf{t}, k') \in \mathcal{I}, \forall k' \in \mathbb{Z}$.

Limit-closed interpretations containing at least one limit fact are infinite in general. Therefore, we use the notion of *pseudointerpretations* which allow for a finite representation of interpretations and correspond naturally to limit-closed interpretations. In Datalog it is common to consider groundings of variables and programs for simplicity, i.e., eliminate variables from rules and programs. But since in Datalog$_\mathbb{Z}$ numeric values range over $\mathbb{Z}$, groundings can be infinite. Kaminski et al. propose the notion of semi-groundings to tackle this issue by eliminating only variables that do not occur in limit atoms. In the following we usually consider semi-ground programs.

In order to express the notion of limit interpretations in a finite way, the authors resort to introducing a new symbol, ∞, in interpretations s.t. for all limit facts $B(\mathbf{t}, k_1), B(\mathbf{t}, k_2) \in \mathcal{J}$: $k_1 = k_2$ for $k_1, k_2 \in \mathbb{Z} \cup \{\infty\}$. Such interpretations over $\mathbb{Z} \cup \{\infty\}$ are called *pseudointerpretations.*

Example 2.6.2. (Interpretations & Pseudointerpretations) Assume $\mathcal{I}$ is a limit-closed interpretation, $\mathcal{I} = \{A(\mathbf{t}, k), B(\mathbf{s}, l), C(4)\}$ for $k \geq 3, l \in \mathbb{Z}$, for some limit Datalog$_\mathbb{Z}$ program, where A is a max predicate, B is a min predicate and $C(2)$ is an exact numeric predicate. The pseudointerpretation corresponding to $\mathcal{I}$ is $\mathcal{J} = \{A(\mathbf{t}, 3), B(\mathbf{s}, \infty), C(4)\}$.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

It is easy to see that limit-closed interpretations and pseudointerpretations admit a one-to-one correspondence. The notion of the classical immediate consequence operator $T_{\mathcal{P}}$, must be adapted to pseudointerpretations. Checking rule applicability is a little more difficult for limit atoms than for normal atoms since atoms in the interpretation might not fit directly but might still be applicable due to the limit semantics. Therefore, $T_{\mathcal{P}}$ converts each $r \in \mathcal{P}$ into a linear integer constraint s.t. r is applicable if and only if its corresponding constraint has a solution. Then $T_{\mathcal{P}}$ derives the optimal head value according to the integer solution of the rule, i.e., the largest value in all integer solutions for max head atoms and the smallest value for min atoms.

Example 2.6.3. (Rule Applicability) Let $r = A(t_1, x) \wedge (x \geq 0) \rightarrow B(t_1, x + 1)$, where A, B are max limit predicates. Furthermore, let $\mathcal{J} = \{A(t, 1)\}$ be a pseudointerpretation. The corresponding integer constraint $C(r, \mathcal{J}) = (x \geq 0) \wedge (x \leq 1)$ has solutions $\{x \mapsto 0\}$ and $\{x \mapsto 1\}$, hence r is applicable to $\mathcal{J}$. Since B is max, $T_{\mathcal{P}}$ derives $B(t_1, 2)$.

KGKMH show that even for limit $\text{Datalog}_{\mathbb{Z}}$ programs, FACTENT is still undecidable. This is due to the fact that multiplication is allowed in rules which implies that checking rule applicability is equivalent to solving non-linear inequalities over integers. Thus, KGKMH restrict the language to *limit linear* $\text{Datalog}_{\mathbb{Z}}$, essentially disallowing multiplication through restricting programs to linear numeric terms. In limit linear $\text{Datalog}_{\mathbb{Z}}$ in each multiplication at most one argument may contain one variable that does not occur in the rule body in a function-free exact atom. This ensures that each variable occurring in a function-free exact atom can only be matched to facts explicitly mentioned in $\mathcal{P}$. By reduction to validity of Presburger Arithmetic, KGKMH showed that the restriction to linear multiplication leads to decidability, i.e., that limit-linear $\text{Datalog}_{\mathbb{Z}}$ is decidable.

Example 2.6.4. (Limit linear $\text{Datalog}_{\mathbb{Z}}$) Let A, B, C, T be max limit atoms and R a EDB numeric (exact) atom. The following rule is not linear, since both variables in the multiplication term occur in limit body atoms.

$$A(x), B(y) \rightarrow C(x \times y) \tag{2.1}$$

In contrast, the following rules are limit-linear, as in the first rule x is a limit variable and 3 a constant. In the second rule both x is a limit variable and y does not occur in a limit atom.

$$A(x) \rightarrow C(x \times 3) \tag{2.2}$$
$$A(x), R(y) \rightarrow T(x \times y) \tag{2.3}$$

A key element in the complexity proofs of limit $\text{Datalog}_{\mathbb{Z}}$ is the fact that we can bound the magnitudes of integers in limit linear $\text{Datalog}_{\mathbb{Z}}$ programs. The main result of KGKMH is the following Theorem. They prove upper bounds on the pseudomodels and integers in pseudomodels of limit $\text{Datalog}_{\mathbb{Z}}$ programs and Theorem 2.6.5 shows that these bounds are tight by proving matching lower bounds.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

Theorem 2.6.5. For a limit-linear program $\mathcal{P}$ and a fact α, FACTENT is coNEXP-complete in combined complexity and coNP-complete in data complexity.

Remarks. The results of Kaminski et al. allow for decidable and even tractable arithmetic in Datalog$_{\mathbb{Z}}$. However, their fragment was not shown to be extendable to more general forms of aggregation (i.e., they do not include other aggregate functions as sum or count). Furthermore, they use a rather implicit semantic notation, which can be cumbersome to use. We try to tackle these problems by introducing an equivalent purely syntactical formulation in Section 4.2. Moreover, as mentioned above, limit Datalog$_{\mathbb{Z}}$ is too weak for reasoning in KGs.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

CHAPTER 3

Related Work

In this chapter we discuss some scientific background on research areas of interest and state-of-the-art research. We focus on high-level discussions and approaches of relevance to our work. This chapter is organized as follows. We start with a short (historical) background on Knowledge Graphs in Section 3.1. In Section 3.2 we discuss a family of Datalog fragments which constitute the basis for modern reasoning languages. Section 3.3 discusses approaches for arithmetic in Datalog. We will return to these approaches in our final Discussion in Chapter 4.5. Finally, we give a brief background on descriptive complexity theory in Section 3.4.

3.1 Background on Knowledge Graphs

Let us briefly discuss some historical developments around KGs, in order to be able to show the importance of KGs and their far-reaching influence in academia and industry. This is especially intriguing since the research around KGs brings together a broad variety of related research fields. There are several key research areas whose development lead to the creation of Knowledge Graphs [GS21]. Important examples are:

1. Automated Reasoning
2. Search Problems
3. Data Management Systems
4. Knowledge Representation

Research in automated reasoning lead to essential developments, such as the creation of programs for solving problems which require “intelligence” [NSS59], the resolution principle [Rob65], and work on the connection between automated theorem proving and deductive databases [GR68]. Further essential findings were the development of search algorithms (e.g., A*) for large search spaces, which is vital for modern AI applications, the development of systems for semantic information retrieval [GR68], and

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

the creation of database management systems which stems from a line of work initiated by Bachman et al. [Bac09]. Collecting and managing knowledge has been important for the industry especially since the late 1970s and 1980s when expert systems and deductive databases grew more and more interesting for enterprises, e.g., as tools to support decision making on business level using enterprise knowledge. Cornerstones in this area were the development of the relational model for databases and the ER model as graph model of databases [Cod70, Che76]. A breakthrough result by Hayes showed that first-order logic was able to formalize frame networks [Hay77]. Frame networks are represented as nodes and relations and built on rather weak logical foundations. Frame networks were considered to be the first knowledge reasoning and knowledge representation systems [Min19]. Soon logic based programming languages such as Prolog with corresponding knowledge based systems emerged [DBS77]. Brachman and Levesque discussed the direct relation between expressive power and complexity of logic and thus created the foundation for investigations of logics with nice properties. This insight gave rise to description logics and F-logic [BL84]. Deductive databases tackled the problem of unifying logic and relational databases into coherent systems. One of the first such systems was LDL [TZ86]. Furthermore, for deductive database systems Datalog emerged as primary programming language. The further development of deductive database systems lead to great interest in finding well behaving Datalog fragments. Disjunctive Datalog [EGM97], Datalog$^{\pm}$ [CGL09a] and probabilistic Datalog [Fuh95] are famous examples for early findings in this area. The so-called Big Data revolution, starting in the 2000s, lead to the fact that large companies were interested in pushing boundaries on systems that operate on huge amounts of data. These developments were especially relevant in the context of AI, machine learning, and deep learning as statistical methods and deep learning require handling large amounts of data. With the ever growing mass of data and the complementing rise in the complexity of data, the need to maintain knowledge in a scalable way has gained more and more weight. In particular, recent developments in AI research pushed the need for handling huge amounts of data. While the term Knowledge Graph (KG) had previously been mentioned in the literature [Bak87], it was coined by the development of Google's Knowledge Graph [Sin12]. There is no standard definition for the term Knowledge Graph in the literature. In spite of that it is common understanding that on a high level a Knowledge Graph is a large network of entities, properties and relationships between entities [EW16] and allows for the integration of data, knowledge, logic, and statistical methods. Hence KGs can be considered as frameworks that unify the developments of various research areas of the past 60 years.

3.1.1 State-of-the-Art Knowledge Graph Systems

The Vadalog system, which is part of the VADA research project [FGNS16, GPL, KKA^{+}17], is a Knowledge Graph system built on top of the underlying Vadalog language, which is based on Datalog. The most interesting part of Vadalog concerning this work is its logical core, with its underlying Vadalog language and reasoning engine. Further prominent examples for KGs are YAGO [SKW07], Freebase [BEP^{+}08] and DBpedia [ABK^{+}07], amongst others. An example for a practical use case scenario for

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

Knowledge Graphs is close link detection between companies, which is highly relevant for financial institutions. Close link detection ensures that companies that have a close relationship to each other cannot act as guarantors for one another, for instance in the case that one company is a subsidiary of another. Modelling such a scenario in a KG is very straight forward, as companies can be represented as entities in the KG and relationships (e.g., a relationship *owns(x,y)* whenever company x owns company y) between those entities as relationships of the KG [GPS19]. Another example is the detection of how certain control units (e.g., families) influence companies, i.e., checking the total number of shares of an asset owned by members of the same unit (family). This use case was denoted *Detection of Family Owned Business* in [BFGS19]. In Section 4.4 we provide an example written in our new language, which can easily express this exemplary use case.

3.2 Datalog$^{\pm}$

Datalog$^{\pm}$ [CGL09b] is a family of logical languages based on Datalog. Additional features, that Datalog$^{\pm}$ languages extend Datalog with, are (one or a combination of) existentials, equality and the logical constant falsum, $\bot$, in rule heads. A naive combination of only one of these features with Datalog immediately leads to undecidability of the language. Hence, languages of the Datalog$^{\pm}$ family introduce syntactic restrictions in order to obtain decidability and in the further course tractability and efficient reasoning algorithms. In the following section we focus on Datalog$^{\pm}$ languages that allow existentials in rule heads, since this extension is the most relevant one for the rest of our work.

From traditional logic programming and Datalog in particular, we know that for each dataset D and set of rules Σ, there is a unique least minimal model denoted $LHM(D \cup \Sigma)$ (for least Herbrand model). The $LHM(D \cup \Sigma)$ contains all ground atoms s.t. $D \cup \Sigma \models a$ for a fact a. Moreover, $LHM(D \cup \Sigma)$ can be computed by a least fixpoint computation. There are several use cases for which we want to define the existence of variables which do not appear in the universe of the input of the logic program at hand [CGL^{+}10]. For instance:

1. Data Exchange: tasks such as copying data between databases with heterogeneous schemas
2. Ontological Query Answering: specifically for Description Logics [BCM^{+}03]. In ontological query answering we want to formalize ontological knowledge about entities and the existence of relationships between entities of a certain domain
3. Web Data extraction: In order to identify objects on a web page and grouping such matched objects into a coherent, new object, existentials are needed

Rules that can have existentials in the rule heads are called *tuple generating dependencies* (TGDs) in database theory. It is known that fundamental reasoning tasks for TGDs, such as deciding whether $D \cup \Sigma \models a$ for a fact a, are undecidable [BV81]. Let us now discuss several languages of the Datalog$^{\pm}$ family and syntactic restrictions introduced by

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

each of those fragments in order to obtain decidability of basic reasoning tasks. Some of the most prominent examples of Datalog$^{\pm}$ languages in research are

- Guarded Datalog$^{\pm}$
- Linear Datalog$^{\pm}$
- Weakly guarded Datalog$^{\pm}$
- Sticky Datalog$^{\pm}$

We will focus on the first three fragments and refer the reader to [CGP10] for a thorough discussion on sticky Datalog$^{\pm}$ and its complexity. In the following we assume the existence of some schema $\mathcal{S}$ for sets of rules, queries and databases (instances) and omit an explicit declaration in definitions and theorems.

3.2.1 Guarded Datalog$^{\pm}$

The syntactic property introduced in guarded Datalog$^{\pm}$ in order to obtain decidability of reasoning in the language is *guardedness*. This property restricts the rule bodies s.t. there is one atom containing all universally quantified variables of the rule.

Definition 3.2.1. A TGD σ is guarded if it contains a body atom which contains all universally quantified variables of σ. A set of TGDs Σ is guarded if each rule $\sigma \in \Sigma$ is guarded.

Consider the following examples, which highlight the difference between guardedness and non-guarded programs.

Example 3.2.2. (Graph Reachability in a Digraph) Given a digraph represented by the relation $Edge/2$, a unary relation $Special$, and a unary relation $Reachable$. The following program computes the set of vertices reachable, encoded in $Reachable$ from the special vertices in the digraph:

$$S(x) \rightarrow Reachable(x)$$
$$Reachable(x), Edge(x, y) \rightarrow Reachable(y)$$

Example 3.2.3. (Transitive Closure) Let $Edge$ be as in Example 3.2.2 and $Closure/2$ be a predicate. The following program computes the transitive closure encoded in $Closure$ on the graph given by $Edge$:

$$Edge(x, y) \rightarrow Closure(x, y)$$
$$Edge(x, y), Closure(y, z) \rightarrow Closure(x, z)$$

Example 3.2.3 is guarded, as the S predicate in the first rule and the $Edge$ predicate in the second rule act as guards and cover all universally quantified variables in the respective rule. Conversely, Example 3.2.2 is not guarded since there is no atom that covers all universal variables in the second rule. Guarded TGDs are theories in the

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

guarded fragment of FO logic [ANvB98]. Satisfiability of guarded FO sentences is 2·EXP-complete. This bound carries over to TGDs as well. The guardedness condition ensures the existence of tree-like universal models, which is the main aspect in the decidability proof. The complexity of guarded Datalog$^{\pm}$ was investigated in [CGK13]. Cali et al. showed EXP-completeness for programs with atoms of bounded arity and 2·EXP-completeness for guarded Datalog$^{\pm}$. The data complexity of guarded Datalog$^{\pm}$, however, is tractable - as the following theorem shows.

Theorem 3.2.4. Let Σ be a set of guarded Datalog$^{\pm}$ rules, D be an instance and let q be a BCQ. Then deciding whether $D \cup \Sigma \models q$ is P-complete in data complexity.

3.2.2 Linear Datalog$^{\pm}$

A fragment of guarded Datalog$^{\pm}$ is linear Datalog$^{\pm}$, which only allows TGDs with a single IDB body atom. Linear Datalog$^{\pm}$ enjoys the property of being FO rewritable which leads to low data complexity. Extensions of linear Datalog$^{\pm}$ have been studied in [BLMS11]. For instance the first rule in Example 3.2.3 and the first rule in Example 3.2.2 are linear. Intuitively, first-order rewritability reduces the entailment problem to a model checking problem of a FO theory. To be precise, a class $\mathcal{C}$ of TGDs is FO rewritable if for every set $\Sigma \in$ of TGDs and every BCQ q, there exists a FO query q_Σ s.t. it holds that $D \cup \Sigma \models q \iff D \models q_\Sigma$ for every instance D. FO rewritability can be established by leveraging tree-like properties of models of linear Datalog$^{\pm}$, similar to guarded Datalog$^{\pm}$. A well-known result in database theory and logic is that linear TGDs are FO rewritable. As FO query answering is in AC^0 in data complexity [Var95] and linear TGDs are FO rewritable, the following theorem that establishes the complexity of linear TGDs immediately follows.

Theorem 3.2.5. Let Σ be a set of linear TGDs, D a database and q a BCQ. Then deciding whether $D \cup \Sigma \models q$ is in AC^0 in data complexity.

3.2.3 Weakly Guarded Datalog$^{\pm}$

Weakly guarded Datalog$^{\pm}$ is a generalization of guarded Datalog$^{\pm}$ obtained by relaxing the guardedness condition. Its definition is based on weakly acyclic TGDs [FKMP05]. Various extension of this fragment have been studied in [Mar09, GHK^{+}13]. In order to define weakly guardedness we need the definition of affected positions of a schema with respect to a set of TGDs, see Definition 2.5.1.

Definition 3.2.6. Let $\mathcal{S}$ be a relational schema, Σ a set of TGDs over $\mathcal{S}$. A TGD $\sigma \in \Sigma$ is weakly guarded w.r.t. Σ if there exist a body atom of σ that contains all universally quantified variables of σ that appear in affected positions of $\mathcal{S}$ w.r.t. Σ.

Observe that the set of guarded TGDs is a subset of the set of weakly guarded TGDs, hence guarded TGDs are weakly guarded.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

Example 3.2.7. (Weakly Guarded TGDs)

$$P(x, y) \rightarrow \exists z R(y, z)$$
$$R(w, x) \rightarrow P(y, x)$$

The set of affected positions for the rules in Example 3.2.7 is $\{R[2], P[2]\}$ and variable x appears only at affected positions in the second rule. Hence the rules in Example 3.2.7 are weakly guarded.

Example 3.2.8. (Not Weakly Guarded TGDs)

$$P(x, y) \rightarrow \exists z R(y, z)$$
$$R(w, x), P(w, y) \rightarrow P(w, x)$$

In Example 3.2.8 the set of affected positions is $\{R[2], P[2]\}$ The first rule is weakly guarded (and even linear) but the second rule is not weakly guarded, since x and y appear only in affected positions but there is no atom covering both variables. The following Theorem establishes the complexity of weakly guarded TGDs.

Theorem 3.2.9. Let Σ be a set of weakly guarded TGDs, D an instance and q a BCQ. Deciding whether $D \cup \Sigma \models q$ is 2EXP-complete.

3.3 Arithmetic and Aggregation in Datalog

Datalog$_{\mathbb{Z}}$ extends ordinary Datalog with integer arithmetic and comparison predicates. These features are needed for data analytic tasks and the computation of traditional problems in the domain of data analytics such as computing shortest path in a graph. Dantsin et al. however showed that a naive combination of Datalog and arithmetic is undecidable [DEGV01]. In this section we review approaches that try to bring together Datalog and arithmetic or aggregation respectively, other than through naive combination which allows arithmetic function symbols over integers in Datalog programs.

3.3.1 Limit Datalog$_{\mathbb{Z}}$

As we discuss limit Datalog$_{\mathbb{Z}}$ and important technicalities in depth in Chapter 2, we only give a brief overview for completeness' sake at this point. Limit Datalog$_{\mathbb{Z}}$ was introduced by Kaminski et al. in [KGK$^+$17] and allows for decidable arithmetic in positive Datalog over integers. Additionally, Kaminksi et al. have recently investigated limit Datalog and stratified negation [KGK$^+$18] and disjunction and negation [KGKH20]. The main idea is to introduce several semantic restrictions for Datalog$_{\mathbb{Z}}$ in order to obtain decidability. First, they introduce a special notion of predicates (limit predicates) which allow to represent interpretations finitely (since interpretations over integers are infinite in general).

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

The authors remark, however, that this restriction is not yet enough to obtain decidatility of the language, since the free use of multiplication renders the language undecidable. Decidability is obtained by essentially disallowing multiplication, i.e., allowing linear terms only. Finally, an efficient (polynomial time) fragment is defined by further restricting the language through preventing divergence of numeric values during the application of the immediate consequence operator used for reasoning.

3.3.2 Datalog^{FS}

Mazuran et al. (MSZ) proposed Datalog^{FS} in their line of work [MSZ13b, MSZ13a], which lays the formal foundation of DeALS, a Datalog-based system for data analytic tasks developed by Shkapsky et al. in [SYZ15]. Mazuran et al. tackle the problem of using aggregate functions in recursive Datalog rules. Aggregation is needed for complex analytic tasks, but ordinary aggregates violate monotonicity requirements the semantics of (the fixpoint computation) Datalog is built upon. Hence, a simple combination is undesirable. Therefore, they introduce an extension of Datalog, called Datalog^{FS}, which allows so-called frequency support goals in Datalog rules. Frequency support goals enable counting of occurrences of predicates that satisfy conjunctions of goals in rules. MSZ introduce a mechanism for monotonic counting whose semantics can be reduced to standard Horn clauses and which is based on the fact that cumulative counting is monotonic in the lattice of set containment [ZYD$^+$17].

MSZ consider Datalog programs that are function symbol free and stratified w.r.t. negation, i.e., a Datalog program $\mathcal{P}$ whose predicates are partitioned into strata s.t. for each $r \in \mathcal{P}$ the head predicate of r is in a stratum which is higher than or equal to the strata of body predicates in r and strictly higher than the stratum of every negated predicate in r (we give a more technical definition in Chapter 2). Furthermore, MSZ use a differential fixpoint computation, which uses a rewriting to avoid redundant computation of facts, to define the semantics of Datalog^{FS} programs.

Definition 3.3.1. (Syntax) Datalog^{FS} is a function free, negation stratified fragment of Datalog without comparison predicates, extending it with the following constructs:

1. FS goals which are either Running-FS goals or Final-FS goals and
2. multioccuring predicates (FS-assert terms)

such that the following hold:

- the head of a rule r can either be (i) an atom or (ii) a multioccuring predicate, and
- the body or a rule r consists of (i) (negated) atoms and (ii) FS-goals

A running-FS goal is of the form $K_j : [\phi]$, where ϕ is a conjunction of positive atoms and K_j is a constant or variable not occurring in ϕ. The expression ϕ is called *b-expression* (bracket expression). Variables in ϕ that also appear outside of the goal are called global and correspond to universally quantified variables. Variables only occurring in ϕ are local and correspond to existentially quantified variables, following the minimum frequency support constraint set by K_j, meaning that there exists at least K_j distinct occurrences

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

of the atoms in ϕ.
A final-FS goal is of the form $K_j :=![\phi]$, and allows to retrieve the exact number of K_j distinct atoms in ϕ. A multioccurring predicate is of the form $P(x) : K$, where x is a tuple of variables. Multioccurring predicates enable the expression of predicates with support greater than one.

Definition 3.3.1 introduces the syntax of Datalog^{FS}. While disallowing arithmetic and comparisons in the syntax directly, the semantics of Datalog^{FS} is based on a rewriting to Datalog using arithmetic, comparisons, and lists. The semantics of each of the constructs introduced above is defined via a rewriting to a set of Datalog rules. The semantics of final-FS goals is defined via a rewriting to running-FS goals with an additional predicate, hence it does not add to the expressive power directly and can be considered syntactic sugar. Each running-FS goal of a Datalog^{FS} program is rewritten separately into a set of Datalog rules. This rewriting based semantics implies that Datalog^{FS} goals do not violate the monotonicity requirement of Datalog, hence FS-goals are monotonic w.r.t. the usual set containment ordering of Datalog. Thus the use of FS-goals in recursion imposes no problem w.r.t. least fixpoint semantics of Datalog. Let us introduce the rewriting for Running FS-goals as exemplary rewriting proposed by MSZ. The semantics of Running FS-goal $K_j : [\phi(x, y)]$ are defined by a rewriting to the predicate $con_j(K_j, x, \_)$, where $con_j(K_j, x, \_)$ is defined as:

$$\phi(x, y) \rightarrow \mathrm{con}_j(1, x, [y])$$
$$\phi(x, y), \mathrm{con}_j(n, x, t), \mathrm{notin}(y, t), n_1 = n + 1 \rightarrow \mathrm{con}_j(n_1, x, [y|x])$$

Where the predicate notin is defined without negation and can be reused between rewritings of different running-FS goals as the definition is generic and independent of the actual goal that is rewritten:

$$\mathrm{notin}(x, [])$$
$$\mathrm{notin}(x, t), x \neq y \rightarrow \mathrm{notin}(x, [y|t])$$

Hence, notin is a simple check if an element is in a list. The intuitive meaning behind the semantics of a running-FS goal is therefore that the goal is satisfied if there are K_j distinct atoms ϕ.

Consider Example 3.3.2 below. Instead of expressing all five different occurrences of atoms $S(x, y)$ explicitly, we can use a running FS goal.

Example 3.3.2. (Running FS-goal)

$$P(x) \rightarrow R(x)$$
$$5 : [S(x, y)] \rightarrow R(y)$$

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

Expressive Power

The authors show several results concerning the expressive power of Datalog^{FS}. Their main results in this respect state that stratified Datalog^{FS} is not a subset of aggregation-stratified Datalog (D^a) and that negation stratified Datalog with addition (Datalog$^{\neg+}$) is a subset of stratified Datalog^{FS}.

Note that since it was shown that Datalog$^{\neg+}$ can express all computable functions on ordered domains [MS95], their results imply that this property also holds for stratified Datalog^{FS}. Hence stratified Datalog^{FS} can express all computable functions on ordered domains. MSZ remark that since Datalog^{FS} can express integer arithmetic, they allow arithmetic functions as part of the language. Note that this extension of course immediately renders Datalog^{FS} undecidable.

In order to compute a Datalog^{FS} program, a semi naive bottom-up procedure is used. Specifically, a differential fixpoint computation algorithm with some modifications for Datalog^{FS}. The differential computation is a rewriting of recursive rule bodies in order to avoid redundant computations (i.e., avoid recomputing the same facts in each iteration). The ordinary semi naive procedure for Datalog produces new δ rules for each recursive goal in a rule by introducing a new δ predicate for each IDB predicate s.t. each atom with a delta predicate was derived only at the previous step of the semi naive algorithm. This enforces the computation of "new" facts only in each iteration of the algorithm. Instead of producing a delta rule for each recursive goal in a recursive rule, MSZ show that it suffices to differentiate the b-expression only once. Hence one differential rule is needed for a running-FS goal in a recursive rule. MSZ propose a series of rewriting steps for the semi naive evaluation of Datalog^{FS} rules, reducing the differential fixpoint for Datalog^{FS} to that of standard Datalog.

Remark. In terms of applications and usability, the authors highlight with several examples that Datalog^{FS} enables expressing a variety of tasks. For instance, they give Datalog^{FS} programs for Markov chains and reachability in generalized hypergraphs. Note, however, that their language is undecidable in general, and no decidable fragment has been identified.

3.3.3 Ross Sagiv Formalism

Ross and Sagiv (RS) introduced a formalism for aggregation in Datalog in [RS92]. Their ideas are based on a similar notion of iterated minimal models [CH85, Prz88, Naq86]. It is known that stratified aggregation, a restricted form disallowing recursion through aggregation, can be used with the traditional least fixpoint operator from Datalog. For many queries in analytic tasks however, aggregation needs to be used in recursion. RS propose a formalism based on monotonicity assumptions of Datalog programs that allows for aggregate functions that can be used in recursion. They also identify syntactic conditions which imply monotonicity of respective programs and can be checked statically.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

Aggregation in Datalog

RS define their language as Datalog with comparisons and addition and multiplication over $\mathbb{N}$ which may appear in rule bodies only. They extend ordinary Datalog with cost predicates that have arguments ranging over a certain domain that is required to be a complete lattice. In order to avoid inconsistencies, RS define that the cost argument of a cost predicate functionally depends on the other atoms of the predicate. This restriction allows duplicate elimination and enforces that atoms with the same object tuples and different cost arguments cannot be true at the same time.

Example 3.3.3. Assume $A(\mathbf{t}, x)$ is an atom with cost argument x. If we compute the sum for $A(\mathbf{t}, x)$ and $A(\mathbf{t}, 5)$ was derived twice (by separate rules), the sum over x should be 5 not 10, since deriving 10 would be non-monotonic. Clearly, if $A(\mathbf{t}, 5)$ and $A(\mathbf{t}, 3)$ have been derived, the sum should be 8.

This restriction also helps avoiding non-monotonic arithmetic. Note that in general the problem of deciding if a given predicate satisfies a functional dependency is undecidable. The authors give syntactic conditions that establish such functional dependencies for parts of a program that are (mutually) recursive. In order to be able to compute aggregations, RS further extend the syntax of the fragment by aggregation functions.

Definition 3.3.4. Let D be a domain called cost domain and let $M(D)$ denote the class of multisets over D. Let $F : M(D) \mapsto R$. We call F an aggregate function. Aggregate functions can be used in subgoals of the form

$$C = FE : p(X_1, \ldots, X_n, Y_1, \ldots Y_m, E)$$

where p is a cost predicate ranging over D. Variables X_i may appear outside the subgoal and are the grouping variables. Variables Y_i are local variables which may only appear in the aggregate subgoal. Variable E appearing in the cost argument is called multiset variable which is used to form the multiset to which F is applied. Finally, variable C is called aggregate variable and must be different from Y_i and E.

A ground instance of an aggregate subgoal γ is obtained by replacing variables C and X_i in γ with constants. Given an interpretation for p, a ground instance γ is satisfied if and only if $c = F(S)$ where S is the multiset defined as

$$S = \pi_E(\sigma_{X_1=x_1,\ldots,X_n=x_n} P(X_1, \ldots X_n, Y_1, Y_n, E))$$

where x_i are the constants from the grounding of the aggregate subgoal and P is the relation of p according to the given interpretation. Additionally, subgoals with $\overset{r}{=}$ instead of $=$ are introduced, with the difference that a ground instance using $\overset{r}{=}$ is false if the multiset F is applied to is empty. Since it may be the case that atoms are needed but have not been derived yet, RS introduce a way to set default values for cost arguments of cost predicates. This mechanism enables us to set a default value which is then true for the respective predicate. It is reasonable to choose the least value of the respective cost domain as default value for a cost argument.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

Example 3.3.5. The rule $n = sum\ Y : T(t, Y)$ computes the sum over cost argument Y of atoms $T(t, Y)$.

Since we want the aggregate variable to be functionally dependent on the global variables, inconsistencies may arise when atoms are defined in different ways simultaneously, for instance once with sum as aggregation function and once with min. Inconsistencies may also arise when functional dependencies are not kept. Thus, RS introduce cost-consistency which requires that the set of head atoms derived by a single application step of all rules in the program satisfies the required functional dependencies of cost arguments. In order to enforce cost-consistency the syntactic notion of conflict-free programs is introduced. We need to ensure that we enforce that the cost argument is functionally determined by the non-cost arguments of a rule. Furthermore, to avoid inconsistencies, RS further restrict rules whose heads unify s.t. it holds the bodies of a pair of rules whose heads unify cannot be satisfied simultaneously or the unified rules generate identical values for the cost argument when using the same non-cost arguments.

Monotonic Programs

Since programs using aggregation do not necessarily omit a unique minimal model in the sense of traditional logic programming (see Example), RS extend standard notions from logic programming of interpretations and models to aggregate Herbrand interpretations and models. Essentially, the aggregate Herbrand base extends the traditional Herbrand base with interpreted constants of domain for cost arguments of cost predicates of a program. Interpretations are subsets of aggregate Herbrand bases s.t. no two atoms differ only in the cost argument (which enforces a functional dependency from the non-cost arguments to the cost argument) and predicates of atoms are given the standard definition over the respective domain. For cost predicates with default values, it is required that an interpretation assigns a cost value to every instance for the non-cost variables. The authors prove that interpretations preserve the ordering of the cost domain of a program.

Example 3.3.6. The following program demonstrates that the use of aggregates implies that there may exist more than one minimal model for programs using aggregates.

$$
\begin{aligned}
P(a) \\
R(a) \\
1 \overset{r}{=} count : R(x) &\rightarrow P(b) \\
1 \overset{r}{=} count : P(x) &\rightarrow R(b)
\end{aligned}
$$

The minimal models are $\{P(b), P(a), R(a)\}$ and $\{R(b), P(a), R(a)\}$.

RS extend the definition of the ordinary immediate consequence operator accordingly, T_P and show that one can ensure cost consistency using properties of T_P. Furthermore, they show monotonicity of the semantics of a program defined by T_P. It follows by Tarski's

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

Fixpoint Theorem, that a least fixpoint of the operator T_P for monotonic programs exists, similar to traditional logic programs. To be able to guarantee the existence of a unique minimal model of a program, it is essential to ensure that all aggregates are monotonic w.r.t. a partial order and that cost values in rules behave monotonically. As RS showed that the least fixpoint operator exists for monotonic programs, it remains to show how we can syntactically recognize if a program is monotonic or not, which guarantees the existence of a least fixpoint of T_P. Intuitively, monotonicity for aggregate function means that when adding more elements to the multiset S an aggregate function F is computing on, the values of the elements in S can only increase the function value of F. In order to guarantee monotonicity, RS proposed syntactic conditions on programs which imply monotonicity of programs. For instance, the authors assume that rules are *well typed*, meaning that the cost domain with a corresponding partial order is equivalent for each aggregate function with multiset variable S and each cost argument in which S occurs.

Remark. The RS formalism introduced above allows for the use of aggregate functions in Datalog programs. The program is however required to fulfil strict monotonicity restrictions and allows only interpretations of a special kind as discussed above. Furthermore (as mentioned in [ZYD+17]), since different lattices for different aggregates can be used, several problems concerning monotonicity and minimal models as discussed in [Gel93] may occur.

3.4 Descriptive Complexity Theory

Computational complexity usually deals with two main measures: time and space and investigates the complexity of problems with respect to these measures. A seminal result by Fagin [Fag74] began to tie complexity to logic as measure of complexity. Fagin showed that the computational complexity of a problem can be understood as the expressivity of a logical language which is needed to describe the problem. This is considered the birth of descriptive complexity. Fagin's result gave insight behind various other proofs in complexity, for example the Immerman-Szelepcsényi Theorem (non-deterministic space classes are closed under complement) [Imm99]. As relational databases can be seen as (finite) logical structures and query languages are based on extensions of first-order logic, descriptive complexity provides foundation for database theory. This also ties finite model theory, a vital area in mathematical logic, to theoretical computer science. Finite model theory is the study of logic on classes of finite structures and provides a framework for establishing connections between different areas of theoretical computer science such as logic, complexity theory, database theory and verification. One of the main motivations behind considering finite structures are applications in computer science where infinite models are clearly not representable. Together with the study of expressive power of logics on finite structures and investigating connections between logic and asymptotic probabilities, descriptive complexity is a main area of finite model theory [KV96].

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

CHAPTER 4

Main Contributions

In this chapter we present our technical results, thereby answering our motivating research questioned outlined in the introductory chapter. In Section 4.1 we show that a naive extension of powerful KG reasoning languages, specifically the Vadalog language, with arithmetic is not possible due to undecidability of such combinations. On the one hand, these results confirm the intuition that the problem of finding suitable candidates of KG languages with arithmetic is hard and that it is highly non-trivial to find a Datalog fragment that fulfils all desiderata and at the same time has good complexity guarantees. On the other hand, these undecidability results motivate us to find a carefully defined and decidable combination of Vadalog with some form of arithmetic.

Since we observed that the definition of limit arithmetic proposed by Kaminski et al. (KGKMH) is not optimal w.r.t. its usability and extendibility, we define a new syntactic version of limit arithmetic, which we call bound $\text{Datalog}_{\mathbb{Z}}$ - based on limit $\text{Datalog}_{\mathbb{Z}}$ ideas by KGKMH - in Section 4.2. One of the main results of this work is the expressivity result of limit $\text{Datalog}_{\mathbb{Z}}$ in Section 4.3. Since we reuse ideas of KGKMH and their well-behaving limit arithmetic for our new language, we investigate the expressive power of limit $\text{Datalog}_{\mathbb{Z}}$ and prove capture results. This allows us to show interesting connections between arithmetic in logic and computational complexity.

Besides this result in descriptive complexity, the central contribution of this work is the definition of our reasoning language in Section 4.4. The main theorem of this section establishes P-completeness of our language. Finally, in Section 4.5 we give a brief comparison of the main aspects of relevant Datalog fragments that support arithmetic or aggregation with our language.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

4.1 Negative Results

Our first contributions are several negative results. We show that several naive extensions of current approaches are immediately undecidable and hence not a good fit for further investigations or in need of further refinement. Firstly, we consider Warded Datalog$^{\pm}$ extended with arithmetic and comparison atoms corresponding to limit linear Datalog$_{\mathbb{Z}}$ as proposed in [KGK+17] and described above.

Definition 4.1.1. (Limit-linear Warded Datalog$_{\mathbb{Z}}$) Limit linear Warded Datalog$_{\mathbb{Z}}$ = Warded Datalog$^{\pm}$ $\cap$ limit linear Datalog$_{\mathbb{Z}}$.

We present several undecidability results, that show that naive combinations of Warded Datalog$^{\pm}$ and arithmetic are undecidable (as expected). A first trivial result can be obtained from results by Kaminski et al.:

Theorem 4.1.2. For a Warded Datalog$_{\mathbb{Z}}$ program $\mathcal{P}$ and a fact α, Factent is undecidable.

Proof. It is known that Factent in Datalog$_{\mathbb{Z}}$ is undecidable [DEGV01]. What is more, Kaminski et al. showed that Factent is undecidable in Datalog$_{\mathbb{Z}}$ even if only addition is used and each atom has at most one numeric position [KGK+17].
The proof of Theorem 1 in [KGK+17] trivially holds for Warded Datalog$_{\mathbb{Z}}$, since Datalog$_{\mathbb{Z}}$ is a subset of Warded Datalog$_{\mathbb{Z}}$ and all of the rules in the proof by Kaminski et al. fulfil the wardedness condition (Definition 2.5.3). Hence, Factent for Warded Datalog$_{\mathbb{Z}}$ programs is undecidable even if only addition is used and each numeric atom has at most one numeric position. □

When restricting recursion as in Vadalog with piece-wise linearity, a proof of this result is not trivial anymore. The proof in [KGK+17] contains rules that are not piece-wise linear, for instance rule (26). Since the predicate State/2 is mutually recursive with Pos/1 and Tape/2 and rule (26) contains both Tape/2 and Pos/1 in its body and State/2 in its head:

$$Time(x) \wedge State(q, x) \wedge Tape(a, y) \wedge Pos(y) \wedge (x \leq y < x + x) \longrightarrow Tape(a', x + y) \quad (25)$$

$$Time(x) \wedge State(q, x) \wedge Tape(a, y) \wedge Pos(y) \wedge (x \leq y < x + x) \longrightarrow State(q', x + x) \quad (26)$$

$$Time(x) \wedge State(q, x) \wedge Tape(a, y) \wedge Pos(y) \wedge Num(u)... \longrightarrow Pos(u) \quad (27)$$

The corresponding predicate graph is depicted in Figure 4.1. This is a clear violation of piece-wise linearity (Definition 2.5.4), since rule (26) contains two atoms in its body whose predicates are mutually recursive with the predicate in the rule head while piece-wise

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

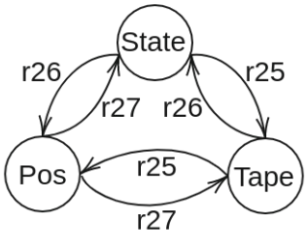

Figure 4.1: Part of the predicate graph for rules 25-27, edge labels indicate the rules which imply the respective edge.

linear rules may at most contain one atom which is mutually recursive with the predicate in the rule head.

We consider the restriction to (piece-wise linear) limit rules as described above:

Theorem 4.1.3. BCQEval for piece-wise linear Warded Datalog$_{\mathbb{Z}}$ programs is undecidable.

Proof. Similar to the proof of Theorem 10 in [KGK$^+$17] we use a reduction from Hilbert's tenth problem [Hil02]. This problem is to decide if a given Diophantine equation has an integer solution, i.e., solving $P(x_1, ...x_n) = 0$, for a multivariate polynomial $P(x_1, ..., x_n)$ with integer coefficients. It is well-known that the problem is undecidable even if the solutions are restricted to be non-negative integers, we use this variant of the problem. For every such polynomial P, let $\mathcal{D} = \{A(0)\}$ and Σ contains the rule:

$$\bigwedge_{i=1}^{n} A(x_i) \wedge P(x_1, ..., x_n) \doteq 0 \longrightarrow B()$$

And $q = B() \longrightarrow Q$. Then $D \cup \Sigma \models q \iff P(x_1, ..., x_n) = 0$ has a non-negative integer solution. Thus, deciding whether $D \cup \Sigma \models q$ is undecidable. □

It is easy to verify that the restriction to limit semantics does not lead to decidability of the problem, thus Factent even for piece-wise linear limit Warded Datalog$_{\mathbb{Z}}$ remains undecidable.

Theorem 4.1.4. BCQEval for piece-wise linear limit Warded Datalog$_{\mathbb{Z}}$ programs is undecidable.

Proof. The proof is equal to the proof of Theorem 4.1.4 with A being a limit min predicate. □

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

4.2 A Syntactic Fragment of Datalog$_\mathbb{Z}$

We consider a syntactic fragment of Datalog$_\mathbb{Z}$, similar to the limit notions of Kaminski et al. in [KGK+17] (which we will refer to as KGKMH). We aim at making the limit semantics explicit through the use of *bound operators*. We first show that we can exploit the techniques of KGKMH for our syntactic fragment, in order to establish fundamental complexity results.

4.2.1 Syntax

We define bound Datalog$_\mathbb{Z}$, which is Datalog$_\mathbb{Z}$ extended with special bound operators *min* and *max* which are used to bound arithmetic expressions and hence integer values in interpretations.

Definition 4.2.1. Bound Datalog$_\mathbb{Z}$ extends the syntax of Datalog$_\mathbb{Z}$ with a set of additional bound operators $\rho(a) \in \{max(a), min(a)\}$ where a is a numeric term and where the following conditions hold:

- Predicates are either object with no numeric position, or numeric;
- Numeric predicates are either exact or bound numeric, where the last position contains a numeric variable ℓ referencing the result of a bound operator ρ, i.e., $\ell = \rho(a)$;
- All exact numeric predicates are EDB;

Atoms with object predicates are object atoms and analogously for other types of predicates. A Datalog$_\mathbb{Z}$ rule r is a bound Datalog$_\mathbb{Z}$ rule if:

- $\mathsf{body}(r) = \emptyset$ or
- each atom in $\mathrm{sb}(r)$ is object, exact numeric or bound atom and $\mathsf{head}(r)$ is an object or bound atom.

A rule in bound Datalog$_\mathbb{Z}$ is of the form:

$$\phi \rightarrow \psi(\mathbf{t}, \rho(a))$$

In database terms: the tuple that is used as grouping argument for ρ (i.e., the tuple for which the bound predicate ρ holds) is determined by the numeric head atom that uses $\ell = \rho(a)$. The *standard body*) ϕ is a conjunction of object and comparison atoms.

We use $\preceq_A$ for $\leq$ ($\geq$) for numeric bound atoms A having a max (min) bound operator in the last position and $\succeq_A, \succ_A, \prec_A$ defined accordingly.

4.2.2 Semantics

For these special operators the group-by argument is determined by the head of the rule. Since the semantics for our special predicates is equivalent to the semantics of limit predicates proposed by KGKMH, we exploit these equivalences and show that their

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

results carry over to our syntactic fragment.
Intuitively, we use bound predicates to only keep the upper and lower bound values of a predicate for a tuple of object, i.e. using a bound predicate $\rho(a)$ means that the value a of ρ for a tuple of objects $\mathbf{t}$ is at least a if ρ is max or at most k if ρ is min. However, we do not enforce the semantics for all predicates. For instance, fact $A(\mathbf{t}, min(3))$ says that A for the tuple of objects $\mathbf{t}$ has value at most 3, as $A(\mathbf{t}, min(3))$ implies that fact $A(\mathbf{t}, min(4))$ also holds. The notions of limit Datalog$_\mathbb{Z}$, such as limit-closed interpretations and corresponding pseudointerpretations, semi-grounding, rule applicability and limit linearness (see Section 2.6) can easily be extended to bound predicates.

Example 4.2.2. Let $\mathcal{P} = \{A(\mathbf{t}, 3), A(\mathbf{s}, 5)\}$. The immediate consequence operator of Datalog would derive: $I_\mathcal{P} = \mathcal{P}$. On the other hand, let $\mathcal{P} = \{A(\mathbf{t}, 3), A(\mathbf{s}, max(5))\}$, then every model must contain $\{A(\mathbf{t}, 3), A(\mathbf{s}, k)\}, \forall k \leq 5$. Thus, we can finitely represent the interpretation with bound facts.

We adapt the notion of limit-closed interpretations from [KGK$^+$17], in order to capture the semantics of the bound operators.

Definition 4.2.3. (Bound-closed Interpretations) An interpretation $\mathcal{I}$ is *bound-closed* if for each bound numeric fact $A(\mathbf{t}, k) \in \mathcal{I}$ with $k = \rho(a)$ it holds that:

- if $\rho = min(a)$ it holds that $\forall k' \geq a : A(\mathbf{t}, min(k')) \in \mathcal{I}$ or
- if $\rho = max(a')$ it holds that $\forall k' \leq a : A(\mathbf{t}, max(k')) \in \mathcal{I}$

An interpretation $\mathcal{I}$ is a model of a program $\mathcal{P}$ if $\mathcal{I} \models \mathcal{P}$ and $\mathcal{I}$ is limit closed. We restrict entailment to only take bound-closed interpretations into account.

4.2.3 Fixpoint of Entailment

The notion of semi-ground rules and semi-groundings can easily be adapted to our fragment. Furthermore, we use the notion of pseudointerpretations to finitely represent interpretations over integers. For bound operators we only need to store the respective bound or ∞ if there is no bound in interpretations.

Definition 4.2.4. A pseudofact is either a fact or of the form $A(\mathbf{t}, \infty)$. A pseudointerpretation $\mathcal{J}$ is a set of pseudofacts s.t. for all bound facts $A(\mathbf{t}, k), A(\mathbf{t}, k') \in \mathcal{J}$ with $k = \rho(a), k' = \rho(a')$ it holds that $a = a'$, for $a, a' \in \mathbb{Z} \cup \{\infty\}$.

It is easy to see that pseudointerpretations and bound-closed interpretations omit a one-to-one correspondence.

Example 4.2.5. Let $\mathcal{I} = \{A(1), A(2), B(\mathbf{t_1}, \ell)\} \cup \{B(\mathbf{t_2}, k)\}$ for all $k \in \mathbb{Z}$ where $\ell = max(5)$ be a bound-closed interpretation. Then the pseudointerpretation $\mathcal{J}$ corresponding to $\mathcal{I}$ is:

$$\{A(1), A(2), B(\mathbf{t_1}, max(5)), B(\mathbf{t_2}, max(\infty))\}$$

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

We adapt the definition of the immediate consequence operator from limit Datalog$_\mathbb{Z}$:

Definition 4.2.6. For a semi-ground program $\mathcal{P}$ and a pseudointerpretation $\mathcal{J}$, the linear integer constraint corresponding to a rule $r \in \mathcal{P}$, $\mathcal{C}(r, \mathcal{J})$ is a conjunction of:

- each comparison atom in $\mathsf{body}(r)$
- $(0 < 0)$ if $\mathsf{body}(r)$ contains
  - an object or exact atom α with $\alpha \notin \mathcal{J}$, or
  - a bound atom $A(\mathbf{t}, \rho(a))$ with $A(\mathbf{t}, k) \notin \mathcal{J}$ for each $k \in \mathbb{Z} \cup \{\infty\}$
- $(s \preceq_A \ell)$ for each $A(\mathbf{t}, \rho(a)), \in \mathsf{body}(r)$ with $A(\mathbf{t}, s) \in \mathcal{J}$ and $s \neq \infty$

Operator $T_\mathcal{P}$ maps $\mathcal{J}$ to the smallest interpretation satisfying $optHead(r, \mathcal{J})$ for each $r \in \mathcal{P}$: $T^0_\mathcal{P} = \emptyset$ and $T^n_\mathcal{P} = T_\mathcal{P}(T^{n-1}_\mathcal{P})$ for $n \geq 1$. Pseudofact $optHead(r, \mathcal{J})$ is obtained as follows:

- $optHead(r, \mathcal{J}) = \mathsf{head}(r)$, if $\mathsf{head}(r)$ is object or exact atom
- $optHead(r, \mathcal{J}) = A(\mathbf{t}, opt(r, \mathcal{J}))$, if $\mathsf{head}(r) = A(\mathbf{t}, \ell)$ with $\ell = min(k)$, where $opt(r, \mathcal{J})$ is the smallest value of ℓ of all solutions to $\mathcal{C}(r, \mathcal{J})$ or ∞ if no such bound exists and dually for $\ell = max(k)$.

4.2.4 Decidability of Bound Datalog$_\mathbb{Z}$

It is easy to see that as with the semantic approach proposed by KGKMH, finite representations of interpretations do not ensure decidability, for completeness' sake we include a formal proof:

Theorem 4.2.7. Factent for positive bound Datalog$_\mathbb{Z}$ is undecidable.

Proof. We reduce Hilbert's tenth problem to the Factent in bound Datalog$_\mathbb{Z}$. For every multivariate polynomial with integer coefficients, $P(x_1, \ldots, x_n)$, let $\mathcal{P}_P$ be the bound Datalog$_\mathbb{Z}$ program consisting of the following rule, where A is a 0-ary object atom:

$$p(x_1, ..., x_n) \doteq 0 \rightarrow A$$

Therefore $\mathcal{P}_P \models A \iff p(x_1, ..., x_n) = 0$ has an integer solution. Thus, Factent is undecidable. □

In order to ensure decidability, we restrict multiplication as in KGKMH, s.t. only linear numeric terms are allowed. Intuitively, in each multiplication, at most one argument is allowed which contains a variable that does not occur in the rule body of a function free, exact atom. Put differently, in each multiplication all arguments must contain only variables which appear in EDB predicates, except for one. Following KGKMH We call this restricted language *linear bound Datalog$_\mathbb{Z}$*. We prove that our syntactic fragment is decidable by reduction to Presburger Arithmetic (PA). As checking validity of a Presburger sentence is known to be decidable, a reduction from Factent to checking validity in PA shows decidability of the former. In order to bound the maximal integer values which is needed for our complexity results, we use the following Lemma.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

Lemma 4.2.8. [KGK+17] Let $\chi = \forall \mathbf{x} \exists \mathbf{y}. \bigvee_{i=0}^{n} \zeta_i$ where each ζ_i be a Presburger sentence where for each i, $|\zeta_i| \leq k$ mentions at most ℓ variables, a is the maximal magnitude of an integer in χ and $m = |\mathbf{x}|$. Then χ is valid if and only if χ is valid over models where each integer variable assumes a value whose magnitude is bounded above by

$$(2^{O(\ell \cdot log \ell)} \cdot a^{k\ell})^{n2^{\ell} \cdot O(m^4)}$$

We adapt techniques from KGKMH in order to obtain complexity results for our syntactic fragment. The proof consists of the following steps.

1. We encode rules of a linear bound Datalog$_\mathbb{Z}$ program $\mathcal{P}$ as conjunction logical sentences PA(r).
2. Then, we show that for a pseudointerpretation $\mathcal{J}$ and a variable assignment μ, $\mathcal{J} \models \sigma \iff \mu \models PA(\sigma)$ for all r.
3. To argue correctness, we show that for a program $\mathcal{P}$, $\mathcal{P} \models \sigma \iff$ there exists a valid Presburger sentence χ.
4. Finally, we show that magnitude of integers in χ can be bounded in order to obtain complexity results.

Note that we slightly extend the signature of Presburger Arithmetic. These extensions can easily be axiomatized using the classical signature of Presburger Arithmetic. The extended signature allows all $i \in \mathbb{Z}$ (as constants $\llcorner i \lrcorner$), multiplication with at least one variable free argument (which can be rewritten as addition) and Boolean variables. We use $\top, \bot$ to denote the Boolean values True and False. We begin with specifying the reduction from linear bound Datalog$_\mathbb{Z}$ to Presburger Arithmetic. Essentially, we encode atoms as variables and rules as Presburger sentences.

Definition 4.2.9. We map predicates to propositional variables: Object predicates A/n to $var_{A\mathbf{t}}$, exact numeric predicates $B/(n+1)$ to $var_{B\mathbf{t}k}$ and numeric bound predicates $C/(n+1)$ to $var_{C\mathbf{t}}$. $fin_{C\mathbf{t}}$ is a propositional variable indicating if the value in C is finite and $val_{C\mathbf{t}}$ is a integer variable capturing the value in C if it is finite. Either it is infinite (∞ in $\mathcal{J}$) or the values are smaller (larger) if C contains max (min).
We reduce a rule σ to σ' as follows: For each atom $\alpha \in \sigma$ we denote its encoding as PA(α)

- If α is a comparison atom: PA(α) = α
- If α is an object atom $A(\mathbf{t})$: PA(α) = $var_{A\mathbf{t}}$
- If α is an exact numeric atom $B(\mathbf{t}, k)$: PA(α) = $var_{B\mathbf{t},k}$
- If α is a numeric bound atom $C(\mathbf{t}, \rho(l))$: PA(α) = $var_{C\mathbf{t}} \wedge (\neg fin_{C\mathbf{t}} \vee l \preceq_C val_{C\mathbf{t}})$ where $\preceq_C$ is $\leq$ ($\geq$) if ρ is max (min).

For a semi-ground program $\mathcal{P}$, PA($\mathcal{P}$) = $\bigwedge_{\sigma \in \mathcal{P}}$ PA(σ) is a FO formula where PA(σ)=$\forall \mathbf{x}.\sigma'$ for all numeric variables $\mathbf{x} \in \sigma$. For a pseudointerpretation $\mathcal{J}$ and a (Boolean and integer) variable assignment μ, $\mathcal{J}$ corresponds to μ if the following conditions hold for all $A, B, C, \mathbf{t}$ and all $k \in \mathbb{Z}$.

- $\mu(var_{A\mathbf{t}}) = \top \iff A(\mathbf{t}) \in \mathcal{J}$

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

- $\mu(var_{B\mathbf{t},k}) = \top \iff B(\mathbf{t},k) \in \mathcal{J}$
- $\mu(var_{C\mathbf{t}}) = \top \iff$ there exists a $k \in \mathbb{Z} \cup \{\infty\}$ s.t. $C(\mathbf{t},\rho(k)) \in \mathcal{J}$
- $\mu(fin_{C\mathbf{t}}) = \top$ and $\mu(val_{C\mathbf{t}}) = k \iff C(\mathbf{t},\rho(k)) \in \mathcal{J}, \forall k \in \mathbb{Z}$

As $k \in \mathbb{Z}$ and $\mu(val_{C\mathbf{t}}) = k$ for some k and $\mathcal{J}$ is a pseudointerpretation (hence cannot contain both $C(\mathbf{t},\rho(\infty))$ and $C(\mathbf{t},\rho(k))$): $C(\mathbf{t},\rho(\infty))$ implies $\mu(fin_{C\mathbf{t}}) = \bot$.

Having the reduction of Definition 4.2.9 at hand, we continue by arguing its correctness.

Lemma 4.2.10. Let $\mathcal{J}$ be a pseudointerpretation and μ a variable assignment corresponding to $\mathcal{J}$. Then it holds that for each ground atom α:

$$\mathcal{J} \models \alpha \iff \mu \models PA(\alpha)$$

Proof. We simply check all possible forms of α:

- α is a comparison atom. $\mathrm{PA}(\alpha) = \alpha$ and its value is independent of $\mathcal{J}$.
- $\alpha = A(\mathbf{t})$ is an object fact. Then $\mathrm{PA}(\alpha) = var_{A\mathbf{t}}$, $\mu(var_{A\mathbf{t}}) = \top \iff A(\mathbf{t}) \in \mathcal{J}$.
- $\alpha = B(\mathbf{t},k)$ is an exact numeric fact. Then $\mathrm{PA}(\alpha) = var_{B\mathbf{t},k}$, thus $\mu(var_{B\mathbf{t},k}) = \top \iff B(\mathbf{t},k) \in \mathcal{J}$.
- $\alpha = C(\mathbf{t},\rho(l))$ is a bound fact. Then $\mathrm{PA}(\alpha) = var_{C\mathbf{t}} \wedge (\neg fin_{C\mathbf{t}} \vee s \preceq_C val_{C\mathbf{t}})$.
  ($\Longrightarrow$) If $\mathcal{J} \models \alpha$, then $C(\mathbf{t},\rho(\infty)) \in \mathcal{J}$ or there exists a $k \in \mathbb{Z}$ s.t. $C(\mathbf{t},\rho(k)) \in \mathcal{J}$ and $l \preceq_C k$. In both cases $\mu(var_{C\mathbf{t}}) = \top$. In the former case $\mu(fin_{C\mathbf{t}}) = \bot$ and in the latter case $\mu(fin_{C\mathbf{t}}) = \top$ and $\mu(val_{C\mathbf{t}}) = k$.
  ($\Longleftarrow$) If $\mu \models PA(\alpha)$ then either $C(\mathbf{t},\rho(\infty)) \in \mathcal{J}$ or there is an integer k s.t. $C(\mathbf{t},\rho(k)) \in \mathcal{J}$. In the former case, $\mu(fin_{C\mathbf{t}}) = \bot$ and in the latter case $\mu(fin_{C\mathbf{t}}) = \top \wedge \mu(val_{C\mathbf{t}}) = k$ implies that $C(\mathbf{t},\rho(k)) \in \mathcal{J}$ with $l \preceq_C k$. □

Corollary 4.2.11. For a semi-ground rule σ: $\mathcal{J} \models \sigma \iff \mu \models PA(\sigma)$

Proof. This follows directly from 4.2.10. □

We extend the base case of Lemma 4.2.10 to bound $\mathrm{Datalog}_{\mathbb{Z}}$ programs.

Theorem 4.2.12. Let $\mathcal{P}$ be a bound linear $\mathrm{Datalog}_{\mathbb{Z}}$ program and a fact α. There exists a Presburger sentence χ s.t. $\mathcal{P} \models \alpha \iff \top \models \chi$

Proof. Corollary 4.2.11 directly implies that

$$\mathcal{P} \models \alpha \iff \models \chi_0 = \forall \mathbf{x}.PA(\alpha) \vee \neg PA(\mathcal{P}) \equiv \forall \mathbf{x}.PA(\mathcal{P}) \implies PA(\alpha),$$

where $\mathbf{x}$ contains the variables defined by the reduction in $\mathrm{PA}(\mathcal{P})$ and $\mathrm{PA}(\alpha)$. Thus $|\mathbf{x}|$ is polynomially bounded by $||\mathcal{P}|| + ||\alpha||$ and the magnitude of each integer in χ_0 is bounded by the maximum magnitude of an integer in $\mathcal{P}$ and α. We now bring ϕ_0 into the desired form by firstly converting each top-level conjunct of $\mathrm{PA}(\mathcal{P})$ into the form $\forall \mathbf{y}_r.\phi_r$ where ϕ_r is CNF for $r \in \mathcal{P}$. Then

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

$$\chi_1 = \forall \mathbf{x}.PA(\alpha) \vee \neg \bigwedge_{r \in \mathcal{P}} \forall \mathbf{y}_r. \bigwedge_{j=1}^{k_r} \phi_r^j,$$

where $n = |\mathcal{P}|$ and k_r is exponentially bounded by $||r||$ and $||\phi_r^j||$ and $|\mathbf{y}_r|$ are linearly bounded by $||r||$. We now can write χ_1 in prenex normal form by moving quantifiers to the front and pushing negations inwards. ψ_r^j is the negation normal form of each χ_r^j:

$$\chi_2 = \forall \mathbf{x} \exists \mathbf{y}.PA(\alpha) \vee \bigvee_{r \in \mathcal{P}} \bigvee_{j=1}^{k_r} \psi_r^j.$$

Then by construction each integer in χ appears in $\mathcal{P}$ or α, $\chi_2 \equiv \chi_0$ by construction and χ_2 is of the required form as each rule semi-ground and linear and the following hold:

- the number of variables in $|\mathbf{y}| = |\bigcup_{r \in \mathcal{P}} y_r|$ is polynomially bounded by $||\mathcal{P}||$ and $|\mathbf{x}|$ is polynomially bounded by $||\mathcal{P}|| + ||\alpha||$
- $||\psi_r^j||$ is linearly bounded by $||\mathcal{P}||$;
- $\sum_{r \in \mathcal{P}}^{n} k_r$ is polynomially bounded by $|\mathcal{P}|$ and exponentially bounded by $\max_{r \in \mathcal{P}} ||r||$;
- the magnitude of each integer in χ_2 is bounded by $\max_{i \in \mathcal{P} \cup \alpha} i$;

By the reduction above we can construct a Presburger sentence χ for a bound linear Datalog$_\mathbb{Z}$ program $\mathcal{P}$ and a fact α that is valid if $\mathcal{P} \models \alpha$. □

Now we can apply Lemma 4.2.8 to the Presburger sentence constructed by the reduction. This provides bounds on the magnitude of integers occurring in models of Presburger sentences of the exact form used in the reduction.

4.2.5 Complexity of Factent in Bound Datalog$_\mathbb{Z}$

Theorem 4.2.13 which also holds in our case, combines the previous lemmata, Lemma 4.2.10 and Lemma, 4.2.8, using the bounds for counter pseudomodel.

Theorem 4.2.13. [KGK+17] For a semi-ground bound linear program $\mathcal{P}$, a dataset $\mathcal{D}$ and a fact α, $\mathcal{P} \cup \mathcal{D} \not\models \alpha$ if and only if there exists a pseudomodel $\mathcal{J}$ of $\mathcal{P} \cup \mathcal{D}$ where $\mathcal{J} \not\models \alpha$, $|\mathcal{J}| < |\mathcal{P} \cup \mathcal{D}|$ and the magnitude of each integer in $\mathcal{J}$ is polynomially bounded by the largest integer in $\mathcal{P} \cup \mathcal{D}$, exponentially by $|\mathcal{P}|$ and double-exponentially by the size of the largest rule $r \in \mathcal{P}$.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

In order to prove lower bounds we can directly apply the following algorithm (Algorithm 4.1) proposed by Kaminski et al. [KGK+17].

Algorithm 4.1: Algorithm for deciding $\mathcal{P} \not\models \alpha$ [KGK+17]

Input: $\mathcal{P}, \alpha$

Result: True iff $\mathcal{P} \not\models \alpha$

1 Compute semi-grounding $\mathcal{P}'$ of $\mathcal{P}$;

2 Non-deterministically guess a pseudointerpretation $\mathcal{J}$ that satisfies the bounds of Theorem 4.2.13;

3 If $T_{\mathcal{P}}(\mathcal{J}) \sqsubseteq \mathcal{J}$, i.e., $\mathcal{J} \models \mathcal{P}'$ and $\mathcal{J} \not\models \alpha$ return True;

As Step 1 and 3 run in exponential time (polynomial in data) and Step 2 runs in non-deterministic exponential time (non-deterministic polynomial in data), Algorithm 4.1 runs in non-deterministic exponential time in combined complexity and non-deterministic polynomial time in data complexity. Hence, Factent $\in$ coNEXP in combined complexity and Factent $\in$ coNP in data complexity for bound Datalog$_{\mathbb{Z}}$ programs.

Lower bounds (i.e., hardness results) follow from reductions from coNP-hard and coNEXP-hard problems (UNSAT and its succinct version, or Square Tiling and its succinct version). The proofs are simple adaptions from [KGK+17, GHK+19].

Proposition 4.2.14. Factent is coNEXP-complete in combined complexity and coNP-complete in data complexity for bound linear Datalog$_{\mathbb{Z}}$ programs.

4.3 Descriptive Complexity Results

In this section we prove results about the expressive power of limit Datalog$_{\mathbb{Z}}$, which can be seen as a contribution to the line of work of Kaminski et al. [KGK+17, KGK+18, KGKH20]. Before we discuss our results in detail, we introduce fundamental techniques from the area of descriptive complexity theory. Firstly, let us introduce some basic notions and standard definitions following [AGK+08]. A vocabulary $\tau = \{R_1, ..., R_n, c_1, ...c_n\}$ is a finite set of relation symbols with specified arity and constant symbols. A τ-structure is a tuple $\mathfrak{A} = (A, R_1^{\mathfrak{A}}, ..., R_n^{\mathfrak{A}}, c_1^{\mathfrak{A}}, ...c_n^{\mathfrak{A}})$ s.t. A is a nonempty set, called the universe of $\mathfrak{A}$. A finite τ-structure has a finite universe. Observe that a relational database is a finite relational structure.

Definition 4.3.1. (Query) Let τ be a vocabulary and k a positive integer.

- A k-ary query Q on a class $\mathcal{K}$ of τ-structures is a mapping with domain $\mathcal{K}$ s.t.
  - For $\mathfrak{A} \in \mathcal{K}$ $Q(\mathfrak{A})$ is a k-ary relation on $\mathfrak{A}$, and
  - Q is closed (preserved) under isomorphism;
- A Boolean query Q is a mapping $Q : \mathcal{K} \mapsto \{0, 1\}$ that is closed preserved under isomorphism. This is equivalent to stating that Q coincides with the subclass

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

$\mathcal{K}' \subseteq \mathcal{K}, \mathcal{K}' = \{\mathfrak{A} \in \mathcal{K} : Q(\mathfrak{A}) = 1\}$. Due to the latter formulation a Boolean query is often said to be a *property* of $\mathcal{K}$.

Definition 4.3.2. (L-definable Query) Let L be a logic and $\mathcal{K}$ a class of τ-structures. A Boolean Query Q on $\mathcal{K}$ is L-definable if there exists an L formula $\varphi(\mathbf{x})$ with free variables $\mathbf{x}$ s.t. for every $\mathfrak{A} \in \mathcal{K}$:

$$Q(\mathfrak{A}) = \{(\mathbf{x}) \in A^k : \mathfrak{A} \models \varphi(\mathbf{a})\}.$$

It is important to say that the notion of L-definability on a class $\mathcal{K}$ of structures is a *uniform* definability notion. Hence, the same L-formula serves as specification of the query on *every* structure in $\mathcal{K}$. This is analogous to the concept of uniform computation of Turing machines known from computational complexity theory [AB09].
It is easy to see that to show that a query Q on $\mathcal{K}$ is L-definable, it suffices to construct a L-formula defining Q on every structure in $\mathcal{K}$. Since our languages are in a Datalog context, let us define the notion of a Datalog query and its complexity.

Definition 4.3.3. (Datalog Query) A Datalog query is a pair (Π, R), of a Datalog program Π and a head predicate $R \in \Pi$. The query (Π, R) associates the result $(\Pi, R)^{\mathfrak{A}}$ which is the interpretation of R computed by Π from the input $\mathfrak{A}$.

A Datalog query essentially consists of a Datalog program and some designated predicate occurring in the program. The result of a query for a certain input structure is obtained by applying the fixpoint semantics of the Datalog language of the query. As in the area of logic programming the data complexity of Datalog (or logics in general) plays a vital role for complexity investigations.

Definition 4.3.4. (Data complexity of Logics) Let L be a logic. The data complexity of L is the family of decision problems Q_φ for each fixed L sentence φ: Given a finite structure $\mathfrak{A}$, does $\mathfrak{A} \models \varphi$?

We next want to define how we can determine whether the data complexity of a logic is in a complexity class $\mathcal{C}$, or even hard for a complexity class $\mathcal{C}$. The following is a standard definition of these two notions.

Definition 4.3.5. (Complexity of Logics) Let L be a logic and $\mathcal{C}$ a complexity class.

- The data complexity of L is in $\mathcal{C}$ if for each L sentence φ, the decision problem Q_φ is in $\mathcal{C}$.
- The data complexity of L is complete for $\mathcal{C}$ if it is in $\mathcal{C}$ and at there exists an L sentence ψ s.t. the decision problem Q_ψ is $\mathcal{C}$-complete

With these definitions at hand, we can formulate our main question of this section. Informally put, from the point of view of descriptive complexity we know that every limit $\text{Datalog}_\mathbb{Z}$ definable query is in coNP and some such queries are even coNP-hard in data

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

complexity by the coNP completeness result of positive limit Datalog$_{\mathbb{Z}}$. The question now is if Datalog$_{\mathbb{Z}}$ is powerful enough to define every coNP computable query. Expressive power of logics is defined via the notion of *capturing* complexity classes.

Definition 4.3.6. (Logics Capturing Complexity Classes) A language $\mathcal{L}$ captures a complexity class $\mathcal{C}$ on a class of databases $\mathcal{D}$ if for each query Q on $\mathcal{D}$ it holds that: Q is definable in $\mathcal{L}$ if and only if it is computable in $\mathcal{C}$.

A cornerstone result in the development of descriptive complexity theory was the following seminal theorem by Fagin showing that the fragment of second-order logic that allows only existential quantification of relation symbols, existential second-order logic ($\exists$SO, sometimes denoted as Σ^1_1), captures non-deterministic polynomial time on the class of all finite structures.

Theorem 4.3.7. (Fagin's Theorem [Fag74]) Let $\mathcal{K}$ be an isomorphism-closed class of finite structures of some finite nonempty vocabulary. Then $\mathcal{K}$ is in NP if and only if $\mathcal{K}$ is definable by an existential second-order sentence.

Fagin's Theorem is a remarkable result, proving the first precise connection between a computational complexity class and logic. Informally it ensures that a property of finite structures is recognizable in non-deterministic polynomial time if it is definable in $\exists$SO. Note that this immediately implies that the universal fragment of second-order logic ($\forall$SO) captures coNP by complementation.

We are now ready to state and prove the main theorem of this section.

Theorem 4.3.8. Let Q be a Boolean query. The following are equivalent:

- Q is computable in coNP;
- Q is definable in semi-positive limit Datalog$_{\mathbb{Z}}$;

That is, semi-positive limit Datalog$_{\mathbb{Z}}$ = coNP.

Note that we extend limit Datalog$_{\mathbb{Z}}$ to semi-positive limit Datalog$_{\mathbb{Z}}$, i.e., we allow negation of EDB (i.e., input) predicates in limit Datalog$_{\mathbb{Z}}$. This is analogous e.g. to the capture result by Blass, Gurevich [BG87], and independently by Papadimitriou [Pap85] that Datalog captures P on successor structures. Datalog alone is too weak to capture P, hence in order to capture P computation, negation of input predicates is needed (we discuss this issue in more detail in the proof). We begin with the simple direction, proving that the second item of Theorem 4.3.8 implies the first, which is straightforward knowing that limit Datalog$_{\mathbb{Z}}$ is coNP complete.

Proposition 4.3.9. The positive limit Datalog$_{\mathbb{Z}}$ definable Boolean queries are all computable in coNP.

Proof. We can clearly compute every positive limit Datalog$_{\mathbb{Z}}$ query in coNP i.e., the proposition follows from Theorem 2.6.5. □

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

It remains to show that every Boolean query on a class of finite structures computable in coNP is definable in positive limit $\text{Datalog}_{\mathbb{Z}}$, i.e., that the properties decidable in coNP on finite structures are definable by positive limit $\text{Datalog}_{\mathbb{Z}}$ queries.

Proposition 4.3.10. All Boolean queries computable in coNP are definable by a Boolean $\text{Datalog}_{\mathbb{Z}}$ query.

Proof. We show that every class of structures $\mathcal{K}$ that is recognizable by a coNP Turing machine is definable by a positive limit $\text{Datalog}_{\mathbb{Z}}$ query. To this end, we construct a positive limit $\text{Datalog}_{\mathbb{Z}}$ query Q over a dataset $\mathcal{D}$ defining an input structure $\mathfrak{A}$.

Before giving the proof, we want to point out several (standard but non-trivial) technicalities that originate mainly from the discrepancy between logic and our model of computation.

- Turing machines compute on string encodings of input structures which implicitly provides a linear order on $\mathfrak{A}$ (the elements of the universe of $\mathfrak{A}$). For our proof this is not a problem since we can simply non-deterministically guess a respective linear order. We thus say a TM M decides a class of structures $\mathcal{K}$ if M decides the set of encodings of structures in $\mathcal{K}$. From now on we fix some kind of canonical encoding (enabled through the linear ordering). We use $enc(\mathfrak{A}, <)$ to refer to the set of encodings of a unordered structure $\mathfrak{A}$ having $<$ as linear order on the universe of $\mathfrak{A}$.
- Turing machines can consider each input bit separately, but Datalog programs cannot detect that some atom is not part of input structure. This is due to the fact that negative information is handled via the closed world assumption and the fact that we only represent positive information in databases [DEGV01]. Thus, we need to slightly extend the syntax of the programs we consider by allowing negated EDB predicates in rule bodies. In our proof this is only relevant for the input encoding.

Consider a single-tape coNP TM M' s.t. M' recognizes a class of structures $\mathcal{K}$, i.e., M' accepts $enc(\mathfrak{A}, <)$ if $\mathfrak{A} \in \mathcal{K}$. Let $M = (\Gamma, S, \delta)$ be a single tape, non-deterministic polynomial time TM recognizing the complement of $\mathcal{K}$. That is, M accepts $enc(\mathfrak{A}, <)$ if $\mathfrak{A} \notin \mathcal{K}$ and rejects $enc(\mathfrak{A}, <)$ if $\mathfrak{A} \in \mathcal{K}$. Let n be the cardinality of the input structure of M. We assume w.l.o.g. that M halts in at most n^k steps (for some constant $k > 0$) and that all computation paths of M end in a halting state.

We represent the non-deterministic guesses of M's non-deterministic transition function as binary string over the alphabet $\{0, 1\}^*$. By our assumption that M halts in at most n^k steps, a guess string corresponding to a computation path, π, in the configuration graph of M represents an integer $i \in [0, n^k]$. We use $g(\pi)$ to denote this representation by adding 1 as the most significant bit in order to ensure each number $g(\pi)$ encodes a unique guess string. This mechanism is the key to encoding non-determinism in limit $\text{Datalog}_{\mathbb{Z}}$. A 0 guess at a certain configuration along a path π in the configuration graph of M can be represented by doubling $g(\pi)$, i.e., $g(\pi') = g(\pi) \cdot 2$ and for a 1 guess we double the number and add 1: $g(\pi') = g(\pi) \cdot 2 + 1$. We use $|\pi|$ to denote the length of π,

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

i.e., the number of configurations in the computation path π of the configuration graph of M.

Encoding, Π_M. We construct a positive limit Datalog$_\mathbb{Z}$ query $Q = (\Pi_M, \alpha)$ such that Q evaluates to true on input $enc(\mathfrak{A}, <)$ if and only if $\mathfrak{A} \notin \mathcal{K}$. The limit Datalog$_\mathbb{Z}$ program Π_M consists of the following sets of rules:

- Π_{succ} computing the successor relation on $\mathfrak{A}$,
- Π_{input}: rules describing the input and ensuring that the encoding of $\mathfrak{A}$ is correct,
- Π_{enc}: rules that describe configurations,
- Π_{comp} rules that enforce computation of M, and
- Π_{rej} rules that check rejection, i.e., if *all* computation paths lead to a rejecting state.

Successor Relation, Π_{succ}. As discussed above, we assume the existence of a successor relation $<$ on $\mathfrak{A}$. We let *succ* be the object predicate encoding this relation and let $succ^+$ indicate the transitive closure of *succ*. We use EDB predicates $first$, $next$, and $last$ to encode the relation accordingly.

Input Encoding, Π_{input}. Note that it is not enough to just explicitly list atomic facts defining the input configuration of M for a given input string u. Hence, we encode successor structures (such as our input) s.t. there exist quantifier-free formulae $\beta_u(\mathbf{y})$ s.t. $\mathfrak{A} \models \beta_u(\mathbf{a}) \iff$ the $\mathbf{a}$-th symbol of the input configuration of M for input $enc(\mathfrak{A}, <)$ is u. Let Π_{input} be the Datalog program equivalent to $\beta_u(\mathbf{y})$.

Configuration Encoding, Π_{enc}. Since we assume that M halts after $m = n^k$ steps and therefore uses at most m tape cells, we can encode a configuration of M as a k-tuple of objects from the universe of $\mathfrak{A}$. We encode a configuration C with the following $(2k+1)$-ary max limit predicates. Intuitively these predicates depend on arguments encoding space (a tape cell numbers), time (an integer corresponding to a configuration), and a guess string $g(\pi)$ for a run π of M:

- For each $q \in S$: $head_q(\mathbf{t}, \mathbf{s}, g(\pi))$. For k-tuples over the universe of $\mathfrak{A}$, $\mathbf{t}$ encodes $|\pi| - 1$ and $\mathbf{s}$ encodes the head position;
- For each $u \in \Gamma$: $tape_u(\mathbf{t}, \mathbf{s}, g(\pi))$. For each $i \in [0, m-1]$ where u is the symbol in the i-th tape cell in configuration C, $\mathbf{s}$ is a k-tuple encoding i and $\mathbf{t}$ is as above

Computation, Π_{comp}. To initialize the computation (the initial configuration), we start by encoding the input on the input tape, fill the rest of the tape with blank symbols, encode the head positioned in the left most position, and set the initial state q_{init} as current state. The tuple $\mathbf{z_0}$ denotes a tuple consisting of k values z_0 and $\mathbf{z'_0}$ is a tuple consisting of $k-1$ repetitions of z_0 used for the initial configuration. Rule 4.1 initializes the head in the left most position, initializing *head* with k-tuples with the first symbol from the dataset $\mathcal{D}$. Similarly, Rule 4.2 encodes the input symbols from the input tape

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

as $tape_u$ for $u \in \{0, 1\}$ and Rule 4.3 fills the rest of the tape with blank symbol $\square$. The last argument of all initialized predicates is set to $1 = g(\pi)$ to encode the first, single, initial configuration of M's configuration graph.

$$first(z_0) \rightarrow head_{q_{init}}(\mathbf{z_0}; \mathbf{z_0}; 1) \tag{4.1}$$
$$first(z_0), input_u(x) \rightarrow tape_u(\mathbf{z_0}; \mathbf{z'_0}, x; 1) \tag{4.2}$$
$$first(z_0), last(z_{max}), succ^+(\mathbf{z'_0}, z_{max}, z_{max}; \mathbf{x}) \rightarrow tape_{\square}(\mathbf{z_0}; \mathbf{x}; 1) \tag{4.3}$$

We add the following rules for each $q \in S \setminus S^{acc} \cup S^{rej}$, each $u, v \in \Gamma$ in order to represent an application of the transition function to the current configuration. Depending on the movement of the head, as indicated by the transition function of M, we let $succ'$ denote $succ(\mathbf{x}', \mathbf{x})$ if the head move is L or $succ(\mathbf{x}, \mathbf{x}')$ if the head move is R. Atom $differ(\mathbf{x}, \mathbf{y})$ is used to ensure that variables $\mathbf{y}$ are different from the head position encoded by $\mathbf{x}$. The rules encode a computation step in the configuration graph of M when 0 is the current guess made by the transition function where the head moves from position $\mathbf{x}$ to position $\mathbf{x}'$, leaving unchanged positions $\mathbf{y}$ intact. In order to model the 0 guess we double $g(\pi)$ by computing $m' = m + m$.

$$\begin{aligned} head_q(\mathbf{t}; \mathbf{x}; m), tape_u(\mathbf{t}; \mathbf{x}; m), tape_u(\mathbf{t}, \mathbf{y}; m), \\ succ(\mathbf{t}, \mathbf{t}'), succ'(\mathbf{x}; \mathbf{x}'), differ(\mathbf{x}; \mathbf{y}), (m + m = m') \rightarrow \\ head_{q'}(\mathbf{t}'; \mathbf{x}'; m'), tape_{u'}(\mathbf{t}'; \mathbf{x}; m'), tape_v(\mathbf{t}'; \mathbf{y}; m') \end{aligned} \tag{4.4}$$

Analogously, we add the following rules for each $q \in S \setminus S^{acc} \cup S^{rej}$ and each $u, v \in \Gamma$ when the current guess is 1. We model a 1 guess as explained above by computing $g(\pi') = g(\pi) \cdot 2 + 1 = m + m + 1$.

$$\begin{aligned} head_q(\mathbf{t}; \mathbf{x}; m), tape_u(\mathbf{t}; \mathbf{x}; m), tape_u(\mathbf{t}, \mathbf{y}; m), \\ succ(\mathbf{t}, \mathbf{t}'), succ'(\mathbf{x}; \mathbf{x}'), differ(\mathbf{x}; \mathbf{y}), (m + m + 1 = m') \rightarrow \\ head_{q'}(\mathbf{t}'; \mathbf{x}'; m'), tape_{u'}(\mathbf{t}'; \mathbf{x}; m'), tape_v(\mathbf{t}'; \mathbf{y}; m') \end{aligned} \tag{4.5}$$

Rejection Check, Π_{rej}. For each rejecting state $r \in S^{reject}$, we derive the *reject* max limit predicate keeping track of the configuration and corresponding guess string for rejecting configurations. We then propagate this information in a backtracking manner in the configuration graph of M's computation depending on the guess strings for each configuration represented by the variables m and m'.

$$head_r(\mathbf{t}; \mathbf{x}; m) \rightarrow reject(\mathbf{t}; m) \tag{4.6}$$
$$reject(\mathbf{t}'; m'), succ(\mathbf{t}; \mathbf{t}'), (m + m = m') \rightarrow reject(\mathbf{t}; m) \tag{4.7}$$
$$reject(\mathbf{t}'; m'), succ(\mathbf{t}; \mathbf{t}'), (m + m + 1 = m') \rightarrow reject(\mathbf{t}; m) \tag{4.8}$$
$$\tag{4.9}$$

Because of the backpropagation we simply need to check if for the initial configuration the *reject* predicate is derived. It is easy to see that fact *confirm* is derived if and only if all configurations reach a rejecting state.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

$$first(z_0), reject(\mathbf{z_0}; 1) \rightarrow confirm \tag{4.10}$$

The following easy to verify claims argue correctness of the encoding above and follow by construction of Π_M.

Claim 4.3.11. If M rejects $enc(\mathfrak{A}, <)$ then $(\Pi_M, confirm)$ evaluates to true on $(\mathfrak{A}, <)$.

Assume M rejects $enc(\mathfrak{A}, <)$. Per definition of a NP TM, each computation path in the configuration graph of M leads to a rejecting state. By construction of Π_M, for each configuration along a path π in the configuration graph of M, at time $\mathbf{t}$, the atom $reject(\mathbf{t}, m)$ where $m = |\pi| - 1$ is derived. Thus, also for the initial configuration the fact *confirm* is derived.

Claim 4.3.12. If $(\Pi_M, confirm)$ evaluates to true on $(\mathfrak{A}, <)$, then M rejects $enc(\mathfrak{A}, <)$.

Assume $(\Pi_M, confirm)$ evaluates to true on $(\mathfrak{A}, <)$. Then for each rejecting state r and each configuration along a computation path π of M leading to r, the atom $reject(\mathbf{t}; m)$ is derived. But then we can "reconstruct" the configuration graph of M from the predicates derived by Π_M. Since *confirm* was also derived, we can conclude the input in the initial state led to a rejection, so M clearly rejects. □

Together with Fagin's Theorem, Theorem 4.3.8 immediately implies the following Corollary, since the universal fragment of second-order logic, ∀SO, is the complement of ∃SO and therefore captures coNP by Theorem 4.3.7.

Corollary 4.3.13. On finite structures positive limit $\text{Datalog}_{\mathbb{Z}} = \forall\text{SO} = \text{coNP}$.

Let us make several brief remarks on implications and related work of Corollary 4.3.13 around universal SO-logic. Least fixpoint logic (LFP) is of great interest in descriptive complexity. It is well-known that LFP cannot express every P-computable query on finite structures. The Immerman-Vardi Theorem shows that, however, LFP can express all P-computable queries on classes of finite ordered structures [Imm80, Imm86]. A well-studied fragment of LFP is LFP_1 which is an extension of FO logic with least fixpoints of positive formulae without parameters and closure under conjunctions, disjunctions, and existential and universal FO quantification (for a formal discussion we refer the reader to [Kol07]). This language is also referred to as LFP(FO), i.e., least fixpoints of FO formulae by Immerman in [Imm99]. It is well-known that the data complexity of LFP_1 is P-complete. The expressive power of LFP_1 was studied by Kleene and Spector on the class of arithmetic $\mathfrak{N} = (\mathbb{N}, +, \times)$. In the seminal *Kleene-Spector Theorem* [Kle55, Gar69], they show that A relation $R \subseteq \mathbb{N}^k$ is LFP_1 definable on $\mathfrak{N}$ if and only if it is definable by a universal second order formula on $\mathfrak{N}$.

The Kleene-Spector Theorem establishes the equivalence $\text{LFP}_1(\mathfrak{N}) = \forall\text{SO}(\mathfrak{N})$. Later this result was extended to countable structures $\mathfrak{A}$ that have a so-called FO coding

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

machinery [Ric79], i.e., countable structures in which finite sequences of each length can be encoded by elements and decoded in a FO definable way [Kol07].

The language ∀SO was conjectured to properly include NLogSPACE [Wei07]. If this conjecture holds then our result readily implies that positive limit Datalog$_{\mathbb{Z}}$ properly includes NLogSPACE.

Now that we have examined limit Datalog$_{\mathbb{Z}}$ in depth and constructed a syntactic variant of the language, we have a good setup in terms of arithmetic in our language. In order to fulfil all language requirements for KG languages, we need to define an extension with existential quantification in rule heads, which will be covered in the following section.

4.4 Towards Reasoning in KGs: Existentials in Rule Heads

In the previous section we have seen that our syntactic fragment is capable of decidable arithmetic, however for reasoning in KGs we need more advanced features, such as existentials in rule heads. Thus, we want to extend our fragment in this direction. Simply extending bound Datalog$_{\mathbb{Z}}$ with existentials in the rule heads leads to undecidability, as Datalog rules (without arithmetic) only with existentials in the rule heads (i.e., TGDs) are undecidable [BV81], even for a fixed set of rules [CGK13]. A fragment of Datalog with existential rule heads is Warded Datalog$^{\pm}$ (see Section 2.5), for which the problem CQAns is known to be P-complete in data complexity and EXP-complete in combined complexity [BGPS19]. Warded Datalog$^{\pm}$ is the foundation of the Vadalog language.

We extend our bound Datalog$_{\mathbb{Z}}$ fragment with warded existentials in the rule heads. Thus, we leverage techniques from Vadalog in order to handle reasoning with existentials in rule heads and prove termination of programs. Moreover, we further restrict arithmetic in our bound fragment using ideas from limit Datalog$_{\mathbb{Z}}$, in order to avoid divergence of numeric terms. In order to arrive at a formal definition of *Warded Bound Datalog$_{\mathbb{Z}}$* we simply extend Definition 2.5.3 in a straightforward way to Datalog$_{\mathbb{Z}}$ to also consider numeric variables. In our language object and numeric variables are being separated in a sense that predicates have arguments that are either all object or object with the last position being numeric. Object predicates may contain null values and numeric predicates may be exact predicates (EDB) or contain arithmetic expressions which are only allowed in bound operators. Note that this is a very powerful extension of the full Vadalog language, using the full potential of arithmetic in bound Datalog$_{\mathbb{Z}}$ with restrictions to obtain tractability and program termination.

Definition 4.4.1. (Warded Bound Datalog$_{\mathbb{Z}}$) Warded Bound Datalog$_{\mathbb{Z}}$ is Bound Datalog$_{\mathbb{Z}}$ extended with existentials in the rule heads such that the following conditions hold:

1. Existentially quantified variables may only appear in object positions.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

2. Each rule is warded, i.e., the ward may only share harmless (numeric and object) variables with the rest of the body atoms (thus we extend Definition 2.5.3 by numeric variables and atoms accordingly)

Observe that we allow predicates with null values and arithmetic terms in rule heads. The following examples showcase the syntax of our language and its relevance to industrial use-cases from e.g., the financial domain.

Example 4.4.2. The following rules are Warded Bound Datalog$_{\mathbb{Z}}$ rules by Definition 4.4.1.

$$R(s,x), P(\mathbf{t},3) \rightarrow \exists \nu R(\nu,3)$$
$$S(s,y), P(\mathbf{t},z) \rightarrow R(s, max(y+z))$$

The following rule is not:

$$P(\mathbf{t},3) \rightarrow \exists z P(\mathbf{t},z)$$

Example 4.4.3. (Family Ownership) The following example is a main use case in financial KG applications introduced in [BFGS19]. This scenario is relevant when we want to study connections between ownership relations of company shares. Note that we use $families$ in order to model *control units*, hence the example below shows how to study how families (or other larger units) control companies. Under the common assumption that individuals might not act individually per se but rather follow a joint consensus, which is decided upon within a control unit (a family in this case), such scenarios are pertinent in industry. Figure 4.2 shows a conceptual depiction of the knowledge base. The blue part is explicitly given database knowledge, while the implicitly inferred knowledge is represented by the green part. Thus, the figure also highlights how we can use existentials to complete relations in KGs, assuming that the $family$ relation is incomplete or even totally missing.

1. $Person(p, \mathbf{x})$: p is a person with property vector $\mathbf{x}$.
2. $Family(f, p)$: person p belongs to family f.
3. $Asset(a, c, v)$: an asset a of a company c whose (absolute) value is v.
4. $Right(o, a, w)$: o has right on a number w of shares of asset a.
5. $Own(f, a, w)$: a family f owns a number w of shares of asset a.

$$Person(p,\mathbf{x}) \rightarrow \exists f Family(f,p). \tag{4.11}$$
$$Person(p,\mathbf{x}), Person(p',\mathbf{x}), Family(f,p) \rightarrow Family(f,p'). \tag{4.12}$$
$$Right(o,a,w), Asset(a,c,v), Right(c,a',w') \rightarrow Right(o,a', max(w+v+w')). \tag{4.13}$$
$$Right(p,a,w), Family(f,p), Own(f,a,x) \rightarrow Own(f,a,max(w+x)). \tag{4.14}$$

In this example, Rule 4.11 defines implicit knowledge, i.e., that a person with a property vector $\mathbf{x}$ exists, then there exists a family relation which contains p. The property vector

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

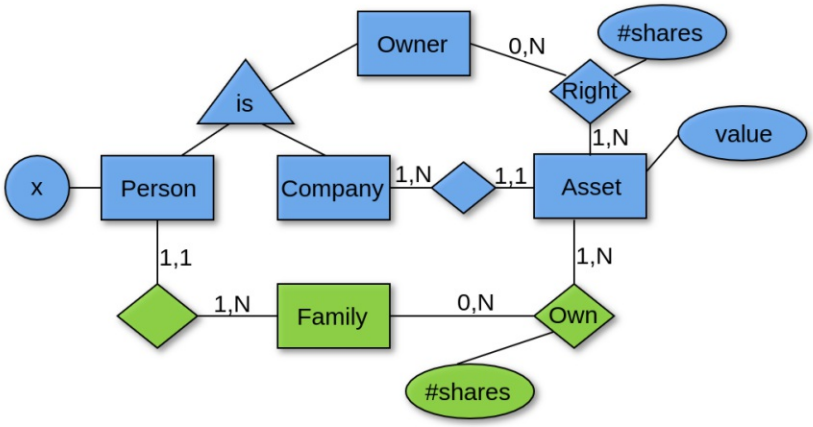

Figure 4.2: Conceptual ER diagram of the family ownership example.

x might for instance be some information identifying a family or, as in [BFGS19], this information might be provided through the use of external functionality, such as a ML based property check computed by an external component of the KG. We omit this advanced functionality here and use the proposed simplification of property vectors that uniquely identify families, because we only want to use constructs from our language, i.e., language constructs for which we have complexity guarantees. Rule 4.12 is used to check whether the families of two persons coincide. Rule 4.13 defines transitivity on the rights of an asset, i.e., situations in which an asset a is defined for some company c, which has rights on another asset, a'. Finally, Rule 4.14 expresses the total ownership of a family based on the rights of its members.

We want to use a chase-like forward chaining operator to define the semantics of Warded Bound Datalog$_\mathbb{Z}$. As we are dealing with a language that supports existential rule heads and arithmetic, we have to make adaptations to the ordinary chase as well as to the applicability check of bound Datalog$_\mathbb{Z}$ in order to support both, arithmetic and existentials. Since we allow arithmetic in our programs, we need to include the applicability check of bound Datalog$_\mathbb{Z}$ programs in the chase algorithm. Finally, we need to handle existentials correctly, otherwise we cannot guarantee termination of the chase procedure.

4.4.1 Algorithm for Deciding Warded Bound Datalog$_\mathbb{Z}$

In this section we present an algorithm for deciding fact entailment for Warded Bound Datalog$_\mathbb{Z}$ programs. This algorithm is the first formal algorithm for a reasoning problem of a decidable Datalog-based language supporting both arithmetic and existential rule heads. We use the termination strategy proposed in [BSG18], since the Vadalog language is the foundation of our fragment. Since our focus is the extension of Vadalog with existentials, we refer the reader to the aforementioned article for an in-depth discussion

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

of the algorithm and use the termination-strategy in a straightforward way in this section while laying the focus on discussing our arithmetic extension.

A simple consequence operator-like procedure for ordinary bound Datalog$_\mathbb{Z}$ arithmetic may not terminate due to potential divergence of arithmetic terms. This would be the case for instance if a program contained the rule $A(max(x)) \rightarrow A(max(x+1))$. Thus, in order to use a chase-like approach, we need to ensure that arithmetic expressions may not diverge by restricting rules to be *stable*, as proposed by Kaminksi et al. (see Definition 4.4.6). However, checking stability is undecidable, hence we define the notion of type consistent programs. Kaminksi et al. showed that type consistency implies stability, while type consistency can be checked efficiently.

Definition 4.4.4. (Type Consistent Program) A warded bound Datalog$_\mathbb{Z}$ rule σ is type consistent if

1. each numeric term is of the form $k_0 + \sum_{i=1}^{n} k_i \times m_i$ where k_0 is an integer and each k_i, $i \in [1, n]$ is a non-zero integer
2. if $\mathsf{head}(\sigma) = A(\mathbf{t}, \rho(\ell))$ then each variable in $\rho(\ell)$ with a positive (negative) coefficient k_i occurs also in a unique bound atom of σ that is of the same (different) type, i.e., min or max, as ρ.
3. for each comparison predicate $(t_1 < t_2)$ or $(t_1 \leq t_2)$ in σ, each variable in t_1 with positive (negative) coefficient also occurs in a unique min (max) operator in a body atom and each variable in t_2 with a positive (negative) coefficient also occurs in max (min) operator in a body atom of σ.

Intuitively, Type Consistency ensures that

1. Each variable contributes to the value of the numeric term it appears in. Hence, multiplication with 0 or subtraction of a term by itself is forbidden;
2. The value in the head containing a numeric variable x behaves analogously to the value in the body occurrence of x, i.e., if the value of x occurring in a rule head increases w.r.t. its occurrence in the body (increases if it appears in max in body and decreases otherwise), then so does the value of the numeric term in the head, and finally
3. Comparisons cannot be invalidated by increasing the values of variables involved.

Note that type consistency is a local and syntactic notion and thus can be checked efficiently. In the following we restrict Warded Bound Datalog$_\mathbb{Z}$ programs to be type-consistent. The next, easy to check proposition follows directly from results by KGKMH.

Proposition 4.4.5. The problem of deciding whether a warded bound program is type-consistent is in LogSPACE.

As mentioned above, in order to use an iterative, chase-like approach, we need to pay attention to diverging arithmetic terms. This is done by keeping track of how arithmetic values are propagated between atoms in rule applications. To keep this

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

mechanism efficient, we need to decide whether a numeric term diverges after a polynomial number of steps. In the context of their work, Kaminski et al. used the notion of *value propagation graphs* based on the values of IP constraints $\mathcal{C}(r, \mathcal{J})$, constructed by the rule applicability check. The value propagation graph $G^{\mathcal{J}}_{\mathcal{P}} = (V, E, \mu)$ of a Warded Bound Datalog$_{\mathbb{Z}}$ program $\mathcal{P}$ w.r.t. an interpretation $\mathcal{J}$ is a digraph containing a unique node $v_{A\mathbf{a}}$ for each bound fact $A(\mathbf{a}, \ell) \in \mathcal{J}$. For each rule σ applicable to $\mathcal{J}$ with $\mathsf{head}(\sigma) = A(\mathbf{a}, \ell_2), v_{A\mathbf{a}} \in V$, $\mathsf{body}(\sigma) = B(\mathbf{b}, \ell_1), v_{B\mathbf{b}} \in V$ and variable m occurring in ℓ, we have $(v_{B\mathbf{b}}, v_{A\mathbf{a}}) \in E$. Finally, we define a function $\mu(e)$ for $e = (v_{B\mathbf{b}}, v_{A\mathbf{a}}) \in E$ establishing edge weights in G, thereby indicating how values are propagated from body atoms to head atoms during evaluation of programs. Function $\mu(e)$ for $e \in E$ is defined as: $\mu(e) = \max\{\mu_r(e) \mid r \in \mathcal{P} \text{ corresponding to } e\}$, where μ_r is defined as:

- $\mu_r(e) = \infty$ if the optimal value of ℓ over $\mathcal{C}(r, \mathcal{J})$ is unbounded;
- $\mu_r(e) = \bot$ if the optimal value of ℓ over $\mathcal{C}(r, \mathcal{J})$ is bounded and $\ell = \infty$;
- $\mu_r(e) = c_a \cdot k - c_b \cdot s$ if the optimal value k of ℓ over $\mathcal{C}(r, \mathcal{J})$ is bounded and $s \in \mathbb{Z}$, where c_a, c_b is 1 if A, B contains max and -1 if A, B contains min.

For instance every edge $e = (v_{B\mathbf{b}}, v_{A\mathbf{a}})$ encodes the information that a rule is applicable to some bound fact $B(\mathbf{b}, \ell) \in \mathcal{J}$, producing the fact $A(\mathbf{a}, \ell + \mu(e))$. The notion of stability allows us to detect divergence of arithmetic terms in Warded Bound Datalog$_{\mathbb{Z}}$ programs. Rule application in a stable program in an iterative manner never decreases the edge weights, hence introducing a form of monotonicity for bound arithmetic.

Definition 4.4.6. (Stability, [KGK$^+$17]) A Warded Bound Datalog$_{\mathbb{Z}}$ program is stable if for all pseudointerpretations $\mathcal{J}, \mathcal{J}'$ with $\mathcal{J} \subseteq \mathcal{J}'$, $G^{\mathcal{J}}_{\mathcal{P}}$ and $G^{\mathcal{J}'}_{\mathcal{P}}$ and each edge e in $G^{\mathcal{J}}_{\mathcal{P}}$:

- $\mu(e) \leq \mu'(e)$;
- $e = (v_{B\mathbf{b}}, v_{A\mathbf{a}})$ and $B(\mathbf{b}, \infty) \in \mathcal{J}$ imply $\mu(e) = \infty$

We are ready to present the main reasoning algorithm, depicted in Algorithm 4.2. Note that this is an extension of the general Vadalog algorithm, extending ideas from Bellomarini et al. in [BSG18] (specifically Algorithm 1 in [BSG18]). In essence, this algorithm is a forward chaining procedure, which derives new facts from the given program step by step in an iterative way. When deriving a new fact γ, we have to pay attention to two things: firstly, we have to pain attention to arithmetic terms and atoms in order to avoid divergence and non-termination. We apply the value propagation graph method proposed by KGKMH in order to keep track of the arithmetic computations. In each iteration of the main loop, we update the value propagation graph (adding new nodes and edges and updating edge weights) of the program w.r.t. the current interpretation $\mathcal{G}^{\mathcal{J}}_{\mathcal{P}}$, which replaces diverging arithmetic terms with ∞ in the interpretation. After this step, we begin to derive new facts by applying the rules (similar to an immediate consequence computation). Thus, for each rule in the input program, we check if its applicable and if so, we try to derive a new fact from it. Because of existential rule heads this step requires an additional termination check since we cannot uncontrollably derive new nulls.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

Therefore, we use the termination check of wardedness (*checkTermination* as proposed in [BSG18]) in order to verify if we can safely derive the current fact. Intuitively, this termination procedure uses several guiding structures, such as warded forests, in order to detect isomorphisms between facts. This mechanism allows us to avoid derivation of superfluous facts. It takes a fact as input and checks, using the underlying data structures, if no isomorphism for the fact to derive exists in the current derivation graph of the program. If so, we add the newly derived fact to the pseudointerpretation $\mathcal{J}_{curr}$ that is currently being computed. Finally, if no new facts have been derived, i.e., the pseudointerpretation that has been computed in the current iteration $\mathcal{J}_{curr}$ is equal to the previously computed one ($\mathcal{J}$), we terminate the reasoning procedure, do a single model check against the fact given as input, and return the result of the check.

Algorithm 4.2: Algorithm for Warded Bound Datalog$_{\mathbb{Z}}$

Input: A type-consistent Warded Bound Datalog$_{\mathbb{Z}}$ program $\mathcal{P} = D \cup \Sigma$, fact α
Result: True if $\mathcal{P} \models \alpha$

```
1  J := ∅;
2  repeat
3  |  J := J_curr;
4  |  update(G_P^J);
5  |  foreach σ ∈ Σ do
6  |  |  if (γ = checkApplicable(σ, J_curr)) != NULL then
7  |  |  |  if checkTermination(γ) then
8  |  |  |  |  J_curr := J_curr ∪ γ;
9  |  |  |  end
10 |  |  end
11 |  end
12 until J_curr = J;
13 return True if J ⊨ α;
```

The subprocedure shown in Algorithm 4.3 represents an important step in rule derivation, namely rule application. Since bound arithmetic with min and max operators is involved, we cannot simply derive a new arithmetic value, but we need to compute the *optimal* value to derive, otherwise it would be possible to derive inconsistent facts which violate the semantics of the min and max bound operators. In essence, Algorithm 4.3 constructs a linear integer constraint which has a solution if a rule is applicable w.r.t. an interpretation, according to the semantics for Warded Bound Datalog$_{\mathbb{Z}}$. Then, there are three cases according to the syntax of Warded Bound Datalog$_{\mathbb{Z}}$:

1. The current rule contains an object or exact numeric head atom, or
2. The current rule contains a bound atom in the head;
3. The current rule contains an existential head;

For Case 1, we simply check if the rule is applicable to the current interpretation and if so, we simply derive the head atom. For Case 2, we need to compute the optimal value of the

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

Algorithm 4.3: Procedure to check and compute rule applicability

Input: Rule σ, pseudo interpretation $\mathcal{J}$
Result: True if σ is applicable to $\mathcal{J}$

```
1  Construct corresponding linear integer constraint C(σ, J);
2  if C(σ, J) has an integer solution then
3  |  if head(σ) is object or exact numeric then
4  |  |  γ_{σ,J} = head(σ);
5  |  end
6  |  else if head(σ) contains a bound operator A(t, ρ(a)) then
7  |  |  let opt(σ, J) be optimal integer solution of C(σ, J);
8  |  |  γ_{σ,J} = A(t, opt(σ, J));
9  |  end
10 |  if head(σ) contains an existential, ∃z A(z, x) then
11 |  |  γ_{σ,J} = A(ν, x) for a new null ν;
12 |  end
13 |  return γ_{σ,J};
14 end
15 else
16 |  return NULL;
17 end
```

linear integer constraint, since we are dealing with min/max bound operators and derive the optimal value. Finally, for Case 3, we need to enforce the semantics of existential rules, construct a fresh, unused null in the object position in case an existential is present and derive an atom containing the new null. The value computed for the current rule $\gamma_{\sigma,\mathcal{J}}$ is then returned, in order to be made available to the main algorithm.

Complexity Analysis

Let us give a formal analysis of our reasoning algorithm introduced above. The following Theorem establishes the complexity of our language.

Theorem 4.4.7. For a Warded Bound Datalog$_{\mathbb{Z}}$, FACTENT is P-complete in data complexity.

Proof. The main aspects for the complexity analysis of our main reasoning algorithm, Algorithm 4.2, are:

1. Line 6 which involves constructing and solving an IP if the body contains numeric atoms, according to the definition of rule applicability of bound Datalog$_{\mathbb{Z}}$ programs;
2. The call in Line 7 which uses the termination-check procedure for the Vadalog language proposed in [BSG18], and
3. Line 4 which updates the value propagation graph as proposed by [KGK$^+$17].

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

Constructing and solving an integer program for the applicability check can be done in polynomial time in data complexity since the integer programs have a fixed number of variables ([KGK+17]). The check-termination procedure requires storing several guiding structures and checking for isomorphism between facts. As shown in [BSG18], for warded forests this can be done in polynomial time. Finally, updating the value propagation graph includes checking whether a node is on a positive weighted cycle in the propagation graph. This check (and update) can be done for instance with a variant of the well-known Floyd-Warshall algorithm in P-time. □

Note that the bound in Theorem 4.4.7 is tight since both reasoning with existentials and arithmetic in stable limit Datalog$_{\mathbb{Z}}$ programs (plain Datalog resp.) is P-complete in data complexity.

4.5 Discussion

In this section we discuss the most crucial differences between our language and existing Datalog languages for data analytic tasks, see Table 4.1 for an overview. Figure 4.3 gives an overview over the complexities of important reasoning languages.

Language	Data Complexity	Arithmetic	KG reasoning
Warded Bound Datalog$_{\mathbb{Z}}$	P-c	**Yes**	**Yes**
Warded Datalog$^{\pm}$	P-c	No	Yes
Limit Datalog$_{\mathbb{Z}}$	coNP-c	Yes	No
Datalog^{FS}	undecidable	Yes	No
RS-formalism	undecidable	Yes	No

Table 4.1: Comparison of Datalog fragments for data analytics. The suffix -c is an abbreviation for "-complete".

While our new language allows for arithmetic and existential rule heads, it partially relies on notions of limit arithmetic introduced by Kaminski et al. We show that we can leverage notions of arithmetic introduced by Kamninski et al. in order to prove decidability and efficient reasoning. The main reason why we extend some of their ideas is that their fragment is the only known decidable Datalog$_{\mathbb{Z}}$ fragment with decidable arithmetic and efficient reasoning. However, since we introduce a different and purely syntactic notion, we succeeded in eliminating obvious problems limit Datalog$_{\mathbb{Z}}$ faces due to its mainly semantic based notion. Firstly, their semantics is rather hard to read for users and therefore also cumbersome to implement. The semantics of programs differs, depending on whether predicates are of min or max semantics, which cannot be discerned when looking at a limit Datalog$_{\mathbb{Z}}$ program, since these are purely semantic restrictions inherent to the respective predicates. We amend this problem by introducing explicitly syntactic min and max bound operators which allows users to instantaneously see whether min or max semantics is applied to an arithmetic term. Furthermore, we argue that

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

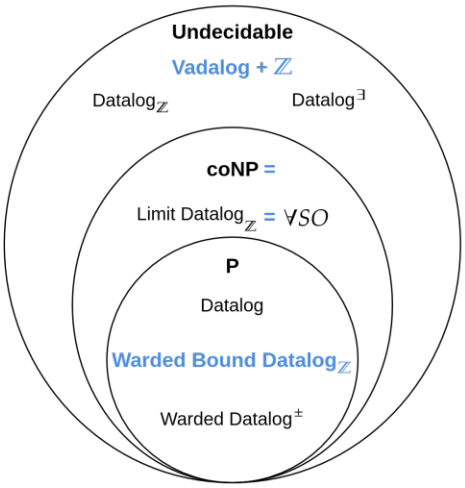

Figure 4.3: Complexities landscape of reasoning languages. $\text{Datalog}_{\mathbb{Z}}$ is Datalog with arithmetic and $\text{Datalog}^{\exists}$ is Datalog with existential rule heads. RS stands for the RS-formalism and ∀SO denotes the universal fragment of second order logic. Our results are highlighted in blue, i.e., the undecidability of Vadalog with unrestricted arithmetic, the equivalence of coNP and limit $\text{Datalog}_{\mathbb{Z}}$, and Warded Bound $\text{Datalog}_{\mathbb{Z}}$ in P.

our syntactical notion is easier to extend to other aggregate functions since it allows for introduction of other syntactic constructs without breaking the min/max semantics of arithmetic terms in bound atoms. Adding aggregate functions such as count or sum in limit $\text{Datalog}_{\mathbb{Z}}$ may not be compatible with "built-in" limit predicates and may require adaptations of their definitions. Furthermore, limit $\text{Datalog}_{\mathbb{Z}}$ is of course too weak for reasoning in KGs since it does not support existential rule heads.

The formalism introduced by Ross and Sagiv (RS) is interesting as, contrary to other results at that time, it deals with all four *common aggregates*, i.e., sum, count, min, and max at once in a coherent manner. A significant drawback of their work is the monotonicity requirement they enforce. Furthermore, their approach allows to use different lattices (acting as cost domains the numeric arguments can range over) for different aggregates. This approach leads to a number of problems, discussed in [Gel93]. Compared with limit $\text{Datalog}_{\mathbb{Z}}$, it supports more aggregate functions directly which is a big benefit. However, both our language and limit $\text{Datalog}_{\mathbb{Z}}$ have better complexity guarantees and are provably decidable, while the RS formalism suffers from the fact that checking monotonicity is undecidable and moreover that monotonicity does not imply decidability of reasoning (of Factent or BCQEval) in general. As other approaches, they do not support rules with existentials in heads and are thus not able to express queries needed for KG reasoning.

Compared to our language (and limit $\text{Datalog}_{\mathbb{Z}}$) the most important downside of the Datalog^{FS} formalism by Mazuran et al. is clearly its undecidability (of Factent).

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

Furthermore, it does not allow existential rule heads and is thus too weak for KG reasoning tasks. However, it is the underlying formalism for the DeALS system [SYZ15], which introduced several optimization techniques in order to facilitate fixpoint computation of Datalog^{FS} programs. Nonetheless the authors do not introduce a decidable fragment and do not provide theoretical complexity proofs of their general language.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

CHAPTER 5

Conclusion

Our main goal was to provide the first complexity result for new Datalog languages that support both existential rule heads and arithmetic. Thereby laying theoretic foundations that show that a combination of arithmetic and advanced reasoning with existential rule heads is in principle possible and even decidable. We not only succeeded in showing that a combination of arithmetic and advanced reasoning in KGs is decidable, but we even proved that it can be done very efficiently in data complexity. Motivated by finding a new, powerful arithmetic extension of KG reasoning languages, we identified the following main gaps in current research:

- There are no results for a combination of TGDs and arithmetic in Datalog;
- In particular, Vadalog has no formal underpinnings for arithmetic;
- Kaminski et al. give the first results for decidable arithmetic in $\text{Datalog}_{\mathbb{Z}}$ but their semantic restrictions make the language hard to use. Also, they do not allow existential rule heads rendering their language inherently to weak for reasoning in KGs;
- There is no expressivity result for decidable arithmetic in Datalog, more specifically for the limit $\text{Datalog}_{\mathbb{Z}}$ fragment.

We closed all of these gaps and provided the first decidability proof of a combination of arithmetic and advanced reasoning in a Datalog language. In Section 4.4 we showed complexity results for Datalog fragment supporting arithmetic and existential quantification powerful enough for complex reasoning tasks, e.g., needed in KGs. In Section 4.3 we proved that the fragment we used to support arithmetic in our language, limit $\text{Datalog}_{\mathbb{Z}}$ as proposed by Kaminski et al., captures coNP.

Possible further directions for future work include the following:

- Investigate other common aggregates such as sum and count and include them in our language;
- Explore further notions of decidable arithmetic in fixpoint logics;

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

- Implement our techniques in the Vadalog system to show practical feasibility of our results;

It would be desirable to also support count and sum aggregates directly in our language since this would allow us to include all common aggregates in one language. Moreover, we believe that there could be other, even more promising notions of decidable arithmetic that can be used in Datalog reasoning. Another interesting direction for future work is to investigate completely new approaches for arithmetic in KG reasoning language that overcome limitations of limit arithmetic. Finally, a practical implementation of our reasoning language in a KG system such as Vadalog would allow us to show empirical results for the efficiency of Warded Bound $\text{Datalog}_{\mathbb{Z}}$ on real-life datasets.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

List of Figures

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

List of Tables

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

List of Algorithms

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Bibliography

[AB09] Sanjeev Arora and Boaz Barak. *Computational Complexity - A Modern Approach*. Cambridge University Press, 2009.

[ABC+11] Foto N. Afrati, Vinayak R. Borkar, Michael J. Carey, Neoklis Polyzotis, and Jeffrey D. Ullman. Map-reduce extensions and recursive queries. In Anastasia Ailamaki, Sihem Amer-Yahia, Jignesh M. Patel, Tore Risch, Pierre Senellart, and Julia Stoyanovich, editors, *EDBT 2011, 14th International Conference on Extending Database Technology, Uppsala, Sweden, March 21-24, 2011, Proceedings*, pages 1–8. ACM, 2011.

[ABK+07] Sören Auer, Christian Bizer, Georgi Kobilarov, Jens Lehmann, Richard Cyganiak, and Zachary G. Ives. Dbpedia: A nucleus for a web of open data. In Karl Aberer, Key-Sun Choi, Natasha Fridman Noy, Dean Allemang, Kyung-Il Lee, Lyndon J. B. Nixon, Jennifer Golbeck, Peter Mika, Diana Maynard, Riichiro Mizoguchi, Guus Schreiber, and Philippe Cudré-Mauroux, editors, *The Semantic Web, 6th International Semantic Web Conference, 2nd Asian Semantic Web Conference, ISWC 2007 + ASWC 2007, Busan, Korea, November 11-15, 2007*, volume 4825 of *Lecture Notes in Computer Science*, pages 722–735. Springer, 2007.

[ACC+10] Peter Alvaro, Tyson Condie, Neil Conway, Khaled Elmeleegy, Joseph M. Hellerstein, and Russell Sears. Boom analytics: exploring data-centric, declarative programming for the cloud. In Christine Morin and Gilles Muller, editors, *European Conference on Computer Systems, Proceedings of the 5th European conference on Computer systems, EuroSys 2010, Paris, France, April 13-16, 2010*, pages 223–236. ACM, 2010.

[AGK+08] Albert Atserias, E. Grädel, P. Kolaitis, L. Libkin, M. Marx, I. Spencer, M. Vardi, Y. Venema, and S. Weinstein. Finite model theory and its applications, springer-verlag. *Comput. Sci. Rev.*, 2(1):55–59, 2008.

[AGP14] Marcelo Arenas, Georg Gottlob, and Andreas Pieris. Expressive languages for querying the semantic web. In Richard Hull and Martin Grohe, editors, *Proceedings of the 33rd ACM SIGMOD-SIGACT-SIGART Symposium on*

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

Principles of Database Systems, PODS'14, Snowbird, UT, USA, June 22-27, 2014, pages 14–26. ACM, 2014.

[ANvB98] Hajnal Andréka, István Németi, and Johan van Benthem. Modal languages and bounded fragments of predicate logic. *J. Philosophical Logic*, 27(3):217–274, 1998.

[AtCG+15] Molham Aref, Balder ten Cate, Todd J. Green, Benny Kimelfeld, Dan Olteanu, Emir Pasalic, Todd L. Veldhuizen, and Geoffrey Washburn. Design and implementation of the logicblox system. In Timos K. Sellis, Susan B. Davidson, and Zachary G. Ives, editors, *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data, Melbourne, Victoria, Australia, May 31 - June 4, 2015*, pages 1371–1382. ACM, 2015.

[Bac09] Charles W. Bachman. The origin of the integrated data store (IDS): the first direct-access DBMS. *IEEE Ann. Hist. Comput.*, 31(4):42–54, 2009.

[Bak87] René Ronald Bakker. *Knowledge Graphs: representation and structuring of scientific knowledge, Ph.D. Thesis, University of Twente, Enschede.* Twente University Press, 1987.

[BCM+03] Franz Baader, Diego Calvanese, Deborah L. McGuinness, Daniele Nardi, and Peter F. Patel-Schneider, editors. *The Description Logic Handbook: Theory, Implementation, and Applications*. Cambridge University Press, 2003.

[BEP+08] Kurt D. Bollacker, Colin Evans, Praveen Paritosh, Tim Sturge, and Jamie Taylor. Freebase: a collaboratively created graph database for structuring human knowledge. In Jason Tsong-Li Wang, editor, *Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2008, Vancouver, BC, Canada, June 10-12, 2008*, pages 1247–1250. ACM, 2008.

[BFGS19] Luigi Bellomarini, Daniele Fakhoury, Georg Gottlob, and Emanuel Sallinger. Knowledge graphs and enterprise AI: the promise of an enabling technology. In *35th IEEE International Conference on Data Engineering, ICDE 2019, Macao, China, April 8-11, 2019*, pages 26–37. IEEE, 2019.

[BG87] Andreas Blass and Yuri Gurevich. Existential fixed-point logic. In Egon Börger, editor, *Computation Theory and Logic, In Memory of Dieter Rödding*, volume 270 of *Lecture Notes in Computer Science*, pages 20–36. Springer, 1987.

[BGPS19] Gerald Berger, Georg Gottlob, Andreas Pieris, and Emanuel Sallinger. The space-efficient core of vadalog. In Dan Suciu, Sebastian Skritek, and Christoph Koch, editors, *Proceedings of the 38th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems, PODS 2019, Amsterdam, The Netherlands, June 30 - July 5, 2019*, pages 270–284. ACM, 2019.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

[BL84] Ronald J. Brachman and Hector J. Levesque. The tractability of subsumption in frame-based description languages. In Ronald J. Brachman, editor, *Proceedings of the National Conference on Artificial Intelligence. Austin, TX, USA, August 6-10, 1984*, pages 34–37. AAAI Press, 1984.

[BLMS11] Jean-François Baget, Michel Leclère, Marie-Laure Mugnier, and Eric Salvat. On rules with existential variables: Walking the decidability line. *Artif. Intell.*, 175(9-10):1620–1654, 2011.

[BNST91] Catriel Beeri, Shamim A. Naqvi, Oded Shmueli, and Shalom Tsur. Set constructors in a logic database language. *J. Log. Program.*, 10(3&4):181–232, 1991.

[BSG18] Luigi Bellomarini, Emanuel Sallinger, and Georg Gottlob. The vadalog system: Datalog-based reasoning for knowledge graphs. *Proc. VLDB Endow.*, 11(9):975–987, 2018.

[BV81] Catriel Beeri and Moshe Y. Vardi. The implication problem for data dependencies. In Shimon Even and Oded Kariv, editors, *Automata, Languages and Programming, 8th Colloquium, Acre (Akko), Israel, July 13-17, 1981, Proceedings*, volume 115 of *Lecture Notes in Computer Science*, pages 73–85. Springer, 1981.

[CGK13] Andrea Calì, Georg Gottlob, and Michael Kifer. Taming the infinite chase: Query answering under expressive relational constraints. *J. Artif. Intell. Res.*, 48:115–174, 2013.

[CGL09a] Andrea Calì, Georg Gottlob, and Thomas Lukasiewicz. Datalog$^{\pm}$: a unified approach to ontologies and integrity constraints. In Ronald Fagin, editor, *Database Theory - ICDT 2009, 12th International Conference, St. Petersburg, Russia, March 23-25, 2009, Proceedings*, volume 361 of *ACM International Conference Proceeding Series*, pages 14–30. ACM, 2009.

[CGL09b] Andrea Calì, Georg Gottlob, and Thomas Lukasiewicz. A general datalog-based framework for tractable query answering over ontologies. In Jan Paredaens and Jianwen Su, editors, *Proceedings of the Twenty-Eigth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, PODS 2009, June 19 - July 1, 2009, Providence, Rhode Island, USA*, pages 77–86. ACM, 2009.

[CGL+10] Andrea Calì, Georg Gottlob, Thomas Lukasiewicz, Bruno Marnette, and Andreas Pieris. Datalog+/-: A family of logical knowledge representation and query languages for new applications. In *Proceedings of the 25th Annual IEEE Symposium on Logic in Computer Science, LICS 2010, 11-14 July 2010, Edinburgh, United Kingdom*, pages 228–242. IEEE Computer Society, 2010.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

[CGP10] Andrea Calì, Georg Gottlob, and Andreas Pieris. Advanced processing for ontological queries. *Proc. VLDB Endow.*, 3(1):554–565, 2010.

[CH85] Ashok K. Chandra and David Harel. Horn clauses queries and generalizations. *J. Log. Program.*, 2(1):1–15, 1985.

[Che76] Peter P. Chen. The entity-relationship model - toward a unified view of data. *ACM Trans. Database Syst.*, 1(1):9–36, 1976.

[CM90] Mariano P. Consens and Alberto O. Mendelzon. Low complexity aggregation in graphlog and datalog. In Serge Abiteboul and Paris C. Kanellakis, editors, *ICDT'90, Third International Conference on Database Theory, Paris, France, December 12-14, 1990, Proceedings*, volume 470 of *Lecture Notes in Computer Science*, pages 379–394. Springer, 1990.

[CMA+12] Neil Conway, William R. Marczak, Peter Alvaro, Joseph M. Hellerstein, and David Maier. Logic and lattices for distributed programming. In Michael J. Carey and Steven Hand, editors, *ACM Symposium on Cloud Computing, SOCC '12, San Jose, CA, USA, October 14-17, 2012*, page 1. ACM, 2012.

[Cod70] E. F. Codd. A relational model of data for large shared data banks. *Commun. ACM*, 13(6):377–387, 1970.

[DBS77] Randall Davis, Bruce G. Buchanan, and Edward H. Shortliffe. Production rules as a representation for a knowledge-based consultation program. *Artif. Intell.*, 8(1):15–45, 1977.

[DEGV01] Evgeny Dantsin, Thomas Eiter, Georg Gottlob, and Andrei Voronkov. Complexity and expressive power of logic programming. *ACM Comput. Surv.*, 33(3):374–425, 2001.

[EGM97] Thomas Eiter, Georg Gottlob, and Heikki Mannila. Disjunctive datalog. *ACM Trans. Database Syst.*, 22(3):364–418, 1997.

[EW16] Lisa Ehrlinger and Wolfram Wöß. Towards a definition of knowledge graphs. In Michael Martin, Martí Cuquet, and Erwin Folmer, editors, *Joint Proceedings of the Posters and Demos Track of the 12th International Conference on Semantic Systems - SEMANTiCS2016 and the 1st International Workshop on Semantic Change & Evolving Semantics (SuCCESS'16) co-located with the 12th International Conference on Semantic Systems (SEMANTiCS 2016), Leipzig, Germany, September 12-15, 2016*, volume 1695 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2016.

[Fag74] Ronald Fagin. Generalized first-order spectra and polynomial-time recognizable sets. *Complexity of computation*, 7:43–73, 1974.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

[FGNS16] Tim Furche, Georg Gottlob, Bernd Neumayr, and Emanuel Sallinger. Data wrangling for big data: Towards a lingua franca for data wrangling. In Reinhard Pichler and Altigran Soares da Silva, editors, *Proceedings of the 10th Alberto Mendelzon International Workshop on Foundations of Data Management, Panama City, Panama, May 8-10, 2016*, volume 1644 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2016.

[FKMP05] Ronald Fagin, Phokion G. Kolaitis, Renée J. Miller, and Lucian Popa. Data exchange: semantics and query answering. *Theor. Comput. Sci.*, 336(1):89–124, 2005.

[FPL11] Wolfgang Faber, Gerald Pfeifer, and Nicola Leone. Semantics and complexity of recursive aggregates in answer set programming. *Artif. Intell.*, 175(1):278–298, 2011.

[Fuh95] Norbert Fuhr. Probabilistic datalog - A logic for powerful retrieval methods. In Edward A. Fox, Peter Ingwersen, and Raya Fidel, editors, *SIGIR'95, Proceedings of the 18th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval. Seattle, Washington, USA, July 9-13, 1995 (Special Issue of the SIGIR Forum)*, pages 282–290. ACM Press, 1995.

[Gar69] Stephen J Garland. C. spector. inductively defined sets of natural numbers. infinitistic methods, proceedings of the symposium on foundations of mathematics, warsaw, 2-9 september 1959, państwowe wydawnictwo naukowe, warsaw, and pergamon press, oxford-london-new york-paris, 1961, pp. 97–102. *The Journal of Symbolic Logic*, 34(2):295–296, 1969.

[Gel92] Allen Van Gelder. The well-founded semantics of aggregation. In Moshe Y. Vardi and Paris C. Kanellakis, editors, *Proceedings of the Eleventh ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, June 2-4, 1992, San Diego, California, USA*, pages 127–138. ACM Press, 1992.

[Gel93] Allen Van Gelder. Foundations of aggregation in deductive databases. In Stefano Ceri, Katsumi Tanaka, and Shalom Tsur, editors, *Deductive and Object-Oriented Databases, Third International Conference, DOOD'93, Phoenix, Arizona, USA, December 6-8, 1993, Proceedings*, volume 760 of *Lecture Notes in Computer Science*, pages 13–34. Springer, 1993.

[GGZ95] Sumit Ganguly, Sergio Greco, and Carlo Zaniolo. Extrema predicates in deductive databases. *J. Comput. Syst. Sci.*, 51(2):244–259, 1995.

[GHK+13] Bernardo Cuenca Grau, Ian Horrocks, Markus Krötzsch, Clemens Kupke, Despoina Magka, Boris Motik, and Zhe Wang. Acyclicity notions for existential rules and their application to query answering in ontologies. *J. Artif. Intell. Res.*, 47:741–808, 2013.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

[GHK+19] Bernardo Cuenca Grau, Ian Horrocks, Mark Kaminski, Egor V. Kostylev, and Boris Motik. Limit datalog: A declarative query language for data analysis. *SIGMOD Rec.*, 48(4):6–17, 2019.

[GOH+13] Birte Glimm, Chimezie Ogbuji, S Hawke, I Herman, B Parisa, A Polleres, and A Seaborne. Sparql 1.1 entailment regimes, 2013. W3C Recommendation 21 March 2013.

[GOP11] Georg Gottlob, Giorgio Orsi, and Andreas Pieris. Ontological queries: Rewriting and optimization. In Serge Abiteboul, Klemens Böhm, Christoph Koch, and Kian-Lee Tan, editors, *Proceedings of the 27th International Conference on Data Engineering, ICDE 2011, April 11-16, 2011, Hannover, Germany*, pages 2–13. IEEE Computer Society, 2011.

[GPL] Georg Gottlob, Norman Paton, and Leonid Libkin. Value added data systems – principles and architecture. `http://vada.org.uk/`. Online; accessed 2020-05-06.

[GPS19] Georg Gottlob, Andreas Pieris, and Emanuel Sallinger. Vadalog: Recent advances and applications. In Francesco Calimeri, Nicola Leone, and Marco Manna, editors, *Logics in Artificial Intelligence - 16th European Conference, JELIA 2019, Rende, Italy, May 7-11, 2019, Proceedings*, volume 11468 of *Lecture Notes in Computer Science*, pages 21–37. Springer, 2019.

[GR68] C. Cordell Green and Bertram Raphael. The use of theorem-proving techniques in question-answering systems. In Richard B. Blue Sr. and Arthur M. Rosenberg, editors, *Proceedings of the 23rd ACM national conference, ACM 1968, USA, 1968*, pages 169–181. ACM, 1968.

[GS21] Claudio Gutiérrez and Juan F. Sequeda. Knowledge graphs. *Commun. ACM*, 64(3):96–104, 2021.

[Hay77] Patrick J. Hayes. In defense of logic. In Raj Reddy, editor, *Proceedings of the 5th International Joint Conference on Artificial Intelligence. Cambridge, MA, USA, August 22-25, 1977*, pages 559–565. William Kaufmann, 1977.

[Hel10] Joseph M. Hellerstein. Datalog redux: experience and conjecture. In Jan Paredaens and Dirk Van Gucht, editors, *Proceedings of the Twenty-Ninth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, PODS 2010, June 6-11, 2010, Indianapolis, Indiana, USA*, pages 1–2. ACM, 2010.

[Hil02] David Hilbert. Mathematical problems. *Bulletin of the American Mathematical Society*, 8(10):437–479, 1902.

[Imm80] Neil Immerman. Upper and lower bounds for first order expressibility. In *21st Annual Symposium on Foundations of Computer Science, Syracuse, New York, USA, 13-15 October 1980*, pages 74–82. IEEE Computer Society, 1980.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

[Imm86] Neil Immerman. Relational queries computable in polynomial time. *Inf. Control.*, 68(1-3):86–104, 1986.

[Imm99] Neil Immerman. *Descriptive complexity*. Graduate texts in computer science. Springer, 1999.

[JK82] David S. Johnson and Anthony C. Klug. Testing containment of conjunctive queries under functional and inclusion dependencies. In Jeffrey D. Ullman and Alfred V. Aho, editors, *Proceedings of the ACM Symposium on Principles of Database Systems, March 29-31, 1982, Los Angeles, California, USA*, pages 164–169. ACM, 1982.

[KGK+17] Mark Kaminski, Bernardo Cuenca Grau, Egor V. Kostylev, Boris Motik, and Ian Horrocks. Foundations of declarative data analysis using limit datalog programs. In Carles Sierra, editor, *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI 2017, Melbourne, Australia, August 19-25, 2017*, pages 1123–1130. ijcai.org, 2017.

[KGK+18] Mark Kaminski, Bernardo Cuenca Grau, Egor V. Kostylev, Boris Motik, and Ian Horrocks. Stratified negation in limit datalog programs. In Jérôme Lang, editor, *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI 2018, July 13-19, 2018, Stockholm, Sweden*, pages 1875–1881. ijcai.org, 2018.

[KGKH20] Mark Kaminski, Bernardo Cuenca Grau, Egor V. Kostylev, and Ian Horrocks. Complexity and expressive power of disjunction and negation in limit datalog. In *The Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2020, The Thirty-Second Innovative Applications of Artificial Intelligence Conference, IAAI 2020, The Tenth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2020, New York, NY, USA, February 7-12, 2020*, pages 2862–2869. AAAI Press, 2020.

[KKA+17] Nikolaos Konstantinou, Martin Koehler, Edward Abel, Cristina Civili, Bernd Neumayr, Emanuel Sallinger, Alvaro A. A. Fernandes, Georg Gottlob, John A. Keane, Leonid Libkin, and Norman W. Paton. The VADA architecture for cost-effective data wrangling. In Semih Salihoglu, Wenchao Zhou, Rada Chirkova, Jun Yang, and Dan Suciu, editors, *Proceedings of the 2017 ACM International Conference on Management of Data, SIGMOD Conference 2017, Chicago, IL, USA, May 14-19, 2017*, pages 1599–1602. ACM, 2017.

[Kle55] Stephen C Kleene. Arithmetical predicates and function quantifiers. *Transactions of the American Mathematical Society*, 79(2):312–340, 1955.

[Kol07] Phokion G Kolaitis. On the expressive power of logics on finite models. In *Finite model theory and its applications*, pages 27–123. Springer, 2007.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

[Kos20] Egor V. Kostylev. Declarative data analysis using limit datalog programs. In Marco Manna and Andreas Pieris, editors, *Reasoning Web. Declarative Artificial Intelligence - 16th International Summer School 2020, Oslo, Norway, June 24-26, 2020, Tutorial Lectures*, volume 12258 of *Lecture Notes in Computer Science*, pages 186–222. Springer, 2020.

[KS91] David B. Kemp and Peter J. Stuckey. Semantics of logic programs with aggregates. In Vijay A. Saraswat and Kazunori Ueda, editors, *Logic Programming, Proceedings of the 1991 International Symposium, San Diego, California, USA, Oct. 28 - Nov 1, 1991*, pages 387–401. MIT Press, 1991.

[KV96] Phokion G. Kolaitis and Moshe Y. Vardi. On the expressive power of variable-confined logics. In *Proceedings, 11th Annual IEEE Symposium on Logic in Computer Science, New Brunswick, New Jersey, USA, July 27-30, 1996*, pages 348–359. IEEE Computer Society, 1996.

[Llo87] John W. Lloyd. *Foundations of Logic Programming, 2nd Edition*. Springer, 1987.

[Mar09] Bruno Marnette. Generalized schema-mappings: from termination to tractability. In Jan Paredaens and Jianwen Su, editors, *Proceedings of the Twenty-Eigth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, PODS 2009, June 19 - July 1, 2009, Providence, Rhode Island, USA*, pages 13–22. ACM, 2009.

[Mar14] Volker Markl. Breaking the chains: On declarative data analysis and data independence in the big data era. *Proc. VLDB Endow.*, 7(13):1730–1733, 2014.

[Min19] Marvin Minsky. *A framework for representing knowledge*. de Gruyter, 2019.

[MMS79] David Maier, Alberto O. Mendelzon, and Yehoshua Sagiv. Testing implications of data dependencies. *ACM Trans. Database Syst.*, 4(4):455–469, 1979.

[MPR90] Inderpal Singh Mumick, Hamid Pirahesh, and Raghu Ramakrishnan. The magic of duplicates and aggregates. In Dennis McLeod, Ron Sacks-Davis, and Hans-Jörg Schek, editors, *16th International Conference on Very Large Data Bases, August 13-16, 1990, Brisbane, Queensland, Australia, Proceedings*, pages 264–277. Morgan Kaufmann, 1990.

[MS95] Inderpal Singh Mumick and Oded Shmueli. How expressive is stratified aggregation? *Ann. Math. Artif. Intell.*, 15(3-4):407–434, 1995.

[MSZ13a] Mirjana Mazuran, Edoardo Serra, and Carlo Zaniolo. A declarative extension of horn clauses, and its significance for datalog and its applications. *Theory Pract. Log. Program.*, 13(4-5):609–623, 2013.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

[MSZ13b] Mirjana Mazuran, Edoardo Serra, and Carlo Zaniolo. Extending the power of datalog recursion. *VLDB J.*, 22(4):471–493, 2013.

[Naq86] Shamim A. Naqvi. A logic for negation in database systems. In Henry F. Korth, editor, *XP / 7.52 Workshop on Database Theory, University of Texas at Austin, TX, USA, August 13-15, 1986*, 1986.

[NSS59] Allen Newell, J. C. Shaw, and Herbert A. Simon. Report on a general problem-solving program. In *Information Processing, Proceedings of the 1st International Conference on Information Processing, UNESCO, Paris 15-20 June 1959*, pages 256–264. UNESCO (Paris), 1959.

[Pap85] Christos H. Papadimitriou. A note the expressive power of prolog. *Bulletin of the EATCS*, 26(21-23):61, 1985.

[Prz88] Teodor C. Przymusinski. On the declarative semantics of deductive databases and logic programs. In Jack Minker, editor, *Foundations of Deductive Databases and Logic Programming*, pages 193–216. Morgan Kaufmann, 1988.

[Ric79] Wayne Richter. Yiannis n. moschovakis. elementary induction on abstract structures. studies in logic and the foundations of mathematics, vol. 77. north-holland publishing company, amsterdam and london, and american elsevier publishing company, inc., new york, 1974, x 218 pp. *Journal of Symbolic Logic*, 44(1):124–125, 1979.

[Rob65] John Alan Robinson. A machine-oriented logic based on the resolution principle. *J. ACM*, 12(1):23–41, 1965.

[RS92] Kenneth A. Ross and Yehoshua Sagiv. Monotonic aggregation in deductive databases. In Moshe Y. Vardi and Paris C. Kanellakis, editors, *Proceedings of the Eleventh ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, June 2-4, 1992, San Diego, California, USA*, pages 114–126. ACM Press, 1992.

[SGL15] Jiwon Seo, Stephen Guo, and Monica S. Lam. Socialite: An efficient graph query language based on datalog. *IEEE Trans. Knowl. Data Eng.*, 27(7):1824–1837, 2015.

[Sin12] Amit Singhal. Introducing the knowledge graph: things, not strings. *Official google blog*, 5:16, 2012.

[SKW07] Fabian M. Suchanek, Gjergji Kasneci, and Gerhard Weikum. Yago: a core of semantic knowledge. In Carey L. Williamson, Mary Ellen Zurko, Peter F. Patel-Schneider, and Prashant J. Shenoy, editors, *Proceedings of the 16th International Conference on World Wide Web, WWW 2007, Banff, Alberta, Canada, May 8-12, 2007*, pages 697–706. ACM, 2007.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

[SYI+16] Alexander Shkapsky, Mohan Yang, Matteo Interlandi, Hsuan Chiu, Tyson Condie, and Carlo Zaniolo. Big data analytics with datalog queries on spark. In Fatma Özcan, Georgia Koutrika, and Sam Madden, editors, *Proceedings of the 2016 International Conference on Management of Data, SIGMOD Conference 2016, San Francisco, CA, USA, June 26 - July 01, 2016*, pages 1135–1149. ACM, 2016.

[SYZ15] Alexander Shkapsky, Mohan Yang, and Carlo Zaniolo. Optimizing recursive queries with monotonic aggregates in deals. In Johannes Gehrke, Wolfgang Lehner, Kyuseok Shim, Sang Kyun Cha, and Guy M. Lohman, editors, *31st IEEE International Conference on Data Engineering, ICDE 2015, Seoul, South Korea, April 13-17, 2015*, pages 867–878. IEEE Computer Society, 2015.

[TZ86] Shalom Tsur and Carlo Zaniolo. LDL: A logic-based data language. In Wesley W. Chu, Georges Gardarin, Setsuo Ohsuga, and Yahiko Kambayashi, editors, *VLDB'86 Twelfth International Conference on Very Large Data Bases, August 25-28, 1986, Kyoto, Japan, Proceedings*, pages 33–41. Morgan Kaufmann, 1986.

[Var95] Moshe Y. Vardi. On the complexity of bounded-variable queries. In Mihalis Yannakakis and Serge Abiteboul, editors, *Proceedings of the Fourteenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, May 22-25, 1995, San Jose, California, USA*, pages 266–276. ACM Press, 1995.

[WBH15] Jingjing Wang, Magdalena Balazinska, and Daniel Halperin. Asynchronous and fault-tolerant recursive datalog evaluation in shared-nothing engines. *Proc. VLDB Endow.*, 8(12):1542–1553, 2015.

[Wei07] Scott Weinstein. Unifying themes in finite model theory. In *Finite Model Theory and Its Applications*, pages 1–25. Springer, 2007.

[ZYD+17] Carlo Zaniolo, Mohan Yang, Ariyam Das, Alexander Shkapsky, Tyson Condie, and Matteo Interlandi. Fixpoint semantics and optimization of recursive datalog programs with aggregates. *Theory Pract. Log. Program.*, 17(5-6):1048–1065, 2017.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN