



Efficient Rule-Based Reasoning and Rules Discovery From Big Data in Many-Valued Settings

DISSERTATION

submitted in partial fulfillment of the requirements for the degree of

Doctor of Philosophy in Computer Science

by

Stefano Sferrazza, MSc.

Student Number 1404770

to the Department of Computer Science

at the University of Oxford

Advisor: Professor Dipl.-Ing. Dr.techn. Georg Gottlob, FRS

Associate Supervisor: Assistant Prof. Dipl.-Ing. Dr.techn. Matthias Lanzinger

The dissertation has been reviewed by:

Professor
Giuseppe De Giacomo

Professor
David Pearce

Oxford, 14th June, 2024

Stefano Sferrazza

ACKNOWLEDGEMENTS

I would like to express my deepest gratitude to the people who have supported me throughout my doctoral journey.

First and foremost, I would like to thank the love of my life, Katia, who has been my constant source of encouragement, support and inspiration in the best and most challenging times. I could not have achieved this goal without her.

I would also like to thank my supervisor, Professor Georg Gottlob, for his invaluable guidance, feedback, and mentorship. He has been an excellent role model and a true leader, who has taught me how to conduct rigorous and impactful research. I am honoured and privileged to have worked with him.

I am also extremely grateful to the associate supervisor, Assistant Professor Matthias Lanzinger, who has provided me with close guidance and assistance throughout the years. He has been very generous with his time and expertise and has helped me improve my research skills and knowledge. He has also been very supportive and encouraging. I have learned much from him, and I treasure his collaboration and friendship.

I would like to thank also Ing. Luigi Bellomarini and Professor Emanuel Sallinger for encouraging me to embark on this wonderful experience.

Finally, a special acknowledgement goes to all the friends who have supported me throughout these years. They filled the past years with laughter and support in the right moments, creating the most beautiful memories I could have ever dreamed of.

ABSTRACT

Abstract

Many critical applications, such as healthcare and security, demand robust decision-making capable of combining expert knowledge with the power of data-driven solutions. However, traditional logic systems often struggle with the inherent uncertainty and multifaceted nature of information in these domains. This thesis addresses this challenge by exploring the integration of symbolic and sub-symbolic reasoning methodologies.

While conventional logic operates with binary truth values (true/false), real-world data is often paired with degrees of truth or uncertainty. This work introduces MV-Datalog and MV-Datalog[±], novel extensions of, respectively, the classic and existential Datalog, specifically designed for reasoning with many-valued data. We introduce the concept of minimal fuzzy models in which each fact assumes the lowest possible truth assignments. This semantic naturally extends the typical minimal model semantics usually employed in logic programming. We prove the existence and efficient computability of unique minimal fuzzy models for MV-Datalog via a linear programming approach. For MV-Datalog[±], we introduce the practice-oriented notion of preferred fuzzy models for efficiently solving data entailment tasks in those settings requiring the use of existential variables. We demonstrate the effectiveness of MV-Datalog by tackling a link detection problem on large-scale, many-valued data.

Moving beyond specific semantics, the thesis also presents a novel optimization technique for reasoning within the Datalog[±] Piece-Wise Linear $\cup$ Warded fragment. We prove that programs in this fragment can be efficiently rewritten into a linear

normal form, where each rule’s body contains at most one intensional atom. We provide an efficient linearization algorithm and extend the theoretical results to allow reusing the linearization algorithm to maintain its beneficial properties in many-valued settings. We explain and show through experiments on synthetic scenarios how the properties of the linearized programs can be used to improve space and time efficiency during CQAns.

This thesis further introduces a patented framework designed to streamline the generation and curation of logical rules by automating the discovery, assessment, and application of association rules within both crisp and many-valued settings. The framework encompasses a suite of methodologies for data preparation, rule mining, evaluation and autonomous application. In addition, the system facilitates the incorporation of expert human reviews and allows for fine-tuning through an extensive array of input parameters. Notably, the mined rules can be used to refine and expand the initial dataset, enhancing the framework’s effectiveness. The practicality and effectiveness of this system are demonstrated through its deployment in a real-world industrial scenario.

Contents

Abstract	v
Contents	vii
1 Introduction	1
2 Background and Related Work	17
2.1 Databases	17
2.2 Graphs and Trees	19
2.3 Datalog	20
2.4 Existential Datalog	25
2.5 Knowledge Graphs (KGs)	43
2.6 Inductive Logic Programming	47
2.7 Fuzzy Logics	52
2.8 Neuro-Symbolic (NeSy) Reasoning	56
3 Reasoning In Many-Valued Settings	65
3.1 MV-Datalog and MV-Datalog [±]	66
3.2 Family Link Detection in Uncertain Settings	83
3.3 Summary	94
4 Optimising Logical Reasoning	97
4.1 Linearizability of PWLW TGDs	102
4.2 Linearizability of fuzzy PWLW TGDs	113
4.3 Implementation and Experiments	120
4.4 Summary	126
5 Automatization Of Datalog Rules Discovery and Application	129
5.1 System Description	132

5.2 An Industry-Led Use Case	164
5.3 Summary	169
6 Conclusions	171
Bibliography	175
Appendix	191
Additional Figures of $ARM(D, I)$	191
Additional Association Rules	196

CHAPTER 1

Introduction

Data is the fuel of the modern world. It is constantly being generated, collected, stored, processed, analyzed, and used for various purposes. It drives innovation, creativity, decision-making, and knowledge discovery in almost every domain, including science, engineering, and business. However, the sole existence of data does not directly translate into value. In order to achieve this, data must be stored, maintained, analyzed, presented and finally interpreted.

Amongst the mentioned tasks, one may argue that the data analysis is the most challenging one. This is due to several data analysis subtasks being neither neutral nor objective. Moreover, analysing data is not a simple or trivial task as it requires appropriate methods, tools, and skills to handle data's complexity, diversity, and volume. Therefore, analysing data quickly, accurately and fairly is essential for ensuring the usefulness, trustworthiness, and ethicality of data analysis.

A framework that allows for a rapid data analysis enables processing and interpreting data in a timely and efficient way, without compromising the accuracy or fairness of the results. This enables quick and informed decision-making, as well as rapid and agile innovation. However, fast data analysis is not easy to achieve, as it requires advanced and scalable algorithms, systems, and architectures that can handle large and complex data in real-time or near real-time.

Accuracy in the data analysis results translates into reliability of the process, as well as confidence and credibility of the results. Accuracy of the analysis is achieved through the minimisation of errors or distortions in the results. Accurate data analysis can improve the quality and performance of the data products, services, and applications other than decision-making made on evidence-based results. However, achieving accurate data analysis is a hard task that requires rigorous and robust methods, tools, and skills to handle data uncertainty, noise, and inconsistency.

Another challenge faced when tackling data analysis problems is the fairness of the analysis. A fair data analysis should interpret data unbiasedly without discriminating or harming any individual or group of data subjects or users. A proper implementation of fair data analysis can ensure the ethicality and responsibility of the data, as well as the respect and protection of the data rights and values. To address this, transparent and careful choices and assumptions are required, as these can affect data representation, interpretation, and impact. Nevertheless, this is by no means trivial, as the pool of data used in the analysis usually contains biases that are difficult to isolate and remove.

The above-presented challenges require many choices which are subjective to the experts conducting the analysis. They must balance and integrate multiple and sometimes conflicting objectives and constraints of data. For example, to achieve accuracy and fairness one must necessarily sacrifice the analysis's speed to perform subtasks such as data cleaning or bias detection. Figure 1.1 shows tasks and subtasks involved in the lifecycle of data. Each of those comes with its own challenges and complexities.

Reasoning in many-valued settings

In the modern era of Big Data, experts must often deal with datasets containing information with (possibly unknown) levels of uncertainty or incompleteness. This constitutes an additional challenge that applications and decision-makers relying on such data must take into account. Uncertainty can arise from various sources, such as Machine Learning (ML), Neural Networks (NN), heuristics-based algorithms, or crowd-sourced data. For example, a database might contain a relation that labels images of animals with the class of animal observed in the image. In modern settings, such labels are commonly derived from the output of ML systems that attempt to classify the image contents. Generally, such systems output possible labels for an image, together with a score that can be interpreted as the level of confidence in the labels' correctness (cf., [122, 103]).

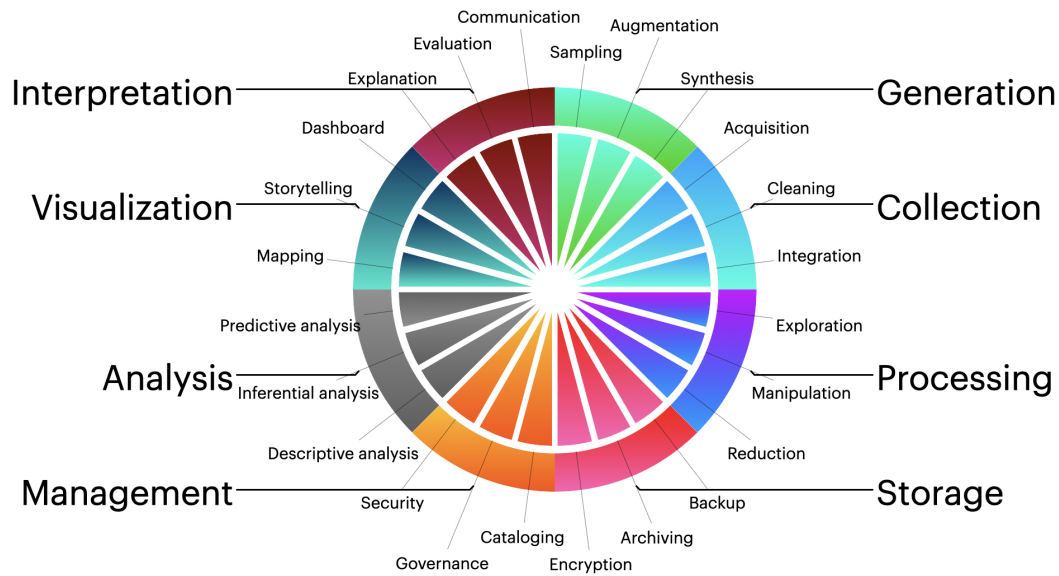


Figure 1.1: Representation of the author's view on the lifecycle of data illustrating the most important tasks and subtasks involved.

Other examples include neural networks that output confidence scores together with their predictions, heuristics providing estimates of the quality of a solution, Internet of Things (IoT) sensors collecting data with measurement errors, or crowd-sourced data containing conflicting or noisy opinions from different users. These sources of uncertainty can be modelled by other means, such as probabilities, fuzzy logic, or belief functions, depending on the inherent nature of the data and its envisioned use.

Suppose to have a database containing incomplete data regarding a specific domain. Let's consider, for example, the Google Knowledge Graph [2] containing data regarding shops (amongst many others). These include companies' attributes such as name, description, website, position, working hours, phone number, etc. Google tries to keep this information as updated as possible using data available online from several sources. However, the level of trust placed in the data sources or even the level of trust placed in the combination of data coming from different sources (possibly containing conflicting values) requires considering the data about companies as not crisp by adding a confidence label to each of the mentioned attributes. Moreover, users are allowed to suggest updates or insertion of missing data. The level of trust posed in crowds of users constitutes an ulterior

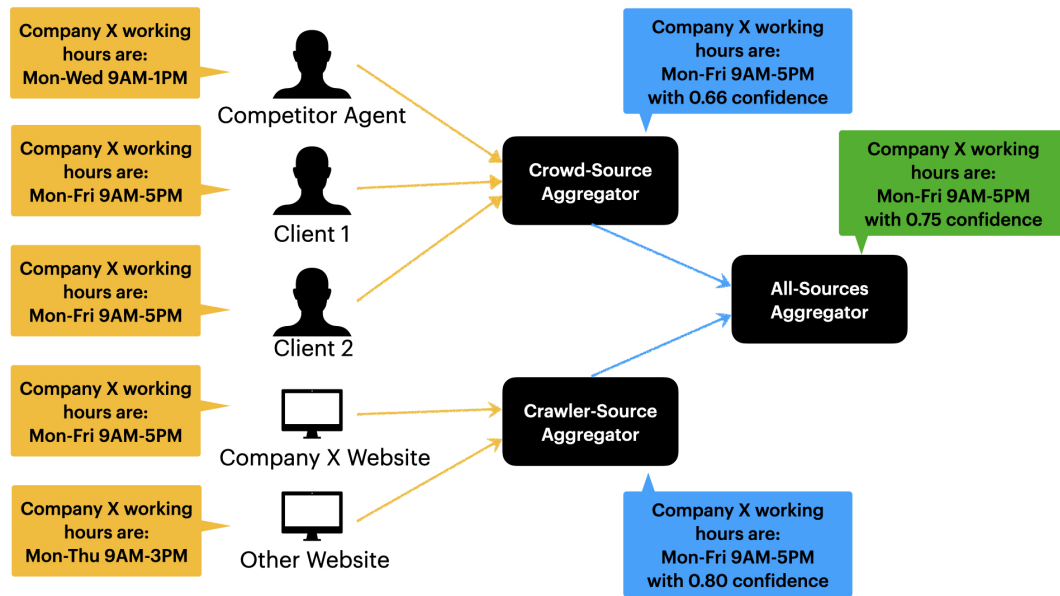


Figure 1.2: Example of a possible aggregation mechanism to manage the aggregation of multiple sources for the same data point. In this example, there are five sources of truth. Two of these, "Competitor Agent" and "Other website", report wrong information, while "Client 1", "Client 2" and "Company X website", report the correct information. The software components "Crowd-Source Aggregator" and "Crawler-Source Aggregator" are responsible for collecting and aggregating data from, respectively, users and websites. A final component named "All-Sources Aggregator" consumes the pairs <data point, confidence> provided by the "Crowd-Source Aggregator" and "Crawler-Source Aggregator" components and produces a single pair <data point, confidence>.

source of uncertainty in the data due to the possible presence of malevolent and/or misinformed agents. Figure 1.2 depicts a scenario where information coming from multiple agents is combined into a single data point paired with a confidence level on the residual truth held by the aggregated data point. When such generated company data is used in further reasoning, it is crucial also to consider the information regarding the truth of the single attributes.

Formal logic constitutes the standard choice to perform symbolic reasoning. However, classical logical reasoning provides no way to consider fuzzy information as they cannot handle the ambiguity and variability of fuzzy data and may produce inaccurate or misleading results. This problem is usually avoided by projecting the

uncertainty levels in the data to crisp true/false values, oftentimes obtained via some simple numerical threshold. Ideally, the level of confidence in the individual data points inferred using uncertain observations should be a function of the uncertainty of the data they depend on. Using a numerical threshold to define whether or not a data point is valid may discard some relevant information that could affect the outcome of the reasoning process or, on the contrary, assume information as entirely true even when it is not. The latter case, when considered in the context of recursive logical rules, could lead to exponential loss of information. Beyond this, thresholding can also lead to problematic outcomes where logical conclusions are true despite their derivation being based on observations made with only moderate confidence. Moreover, applying different thresholds may lead to different conclusions, which may not be compatible. This makes the threshold an ulterior hyperparameter that must be carefully tuned for each application. Consider the following example containing a single logical rule and three data points (aka facts or EDBs):

Example 1.

$$\begin{aligned}
 &rel(X, Y), rel(Y, Z) \rightarrow rel(X, Z). \\
 &0.75rel(A, B). \\
 &0.75rel(B, C). \\
 &\dots \\
 &0.75rel(Y, Z).
 \end{aligned}$$

where *rel* is a predicate indicating the relationship between two users and the number to the left of the facts *rel*(...) represents a certainty associated with such a fact. If a threshold of, say, 0.7 is applied to distinguish true facts from false facts, the transitive closure of *rel*(...) would be computed, with the confidence in providing the fact *rel*(A, C) holding the same value as the confidence for *rel*(A, Z). However, commonsense would suggest that the relationship between A and C should be stronger than between A and Z.

With observations from ML systems becoming a commonplace feature of many modern reasoning settings, the projection to binary truth values severely limits the immense potential of integrating ML observations with logic programming.

Logical languages such as Datalog and its extensions have long been essential languages in databases and at the foundation of many rule-based reasoning formalisms. The Datalog[±] family of languages has rapidly emerged as the *de*

facto standard formalism for expressing logic-based ontological reasoning tasks [12, 23, 39, 164]. Over the last few decades, many Datalog-based systems have been developed. Some examples of these include BOOM [10], DeALS[152], Myria [168], Socialite [148], Overlog [143], Dyna [59], Yedalog [46], RDFox [128] and Vadalog [30]. Moreover, Datalog has been proven to be a good fit for reasoning over Knowledge Graphs (KG) [30, 41].

While there is no unique definition for KGs, in this text, we refer to the following as it has been widely adopted: *a knowledge graph acquires and integrates information into an ontology and applies a reasoner to derive new knowledge* [58]. As this definition highlights, a key feature of KGs is the *integration* of several sources of information into a unified ontology, which can then be used to *derive* novel knowledge.

Intuitively speaking, an ontological reasoning task in Datalog[±] consists of answering a Conjunctive Query (CQ) Q over a database D , augmented with a set of logical rules Σ . Rules in Σ are function-free Horn clauses, potentially including existential quantification, i.e., *Tuple-Generating Dependencies* (TGDs). These follow the form $\forall \mathbf{x} \phi(\mathbf{x}) \rightarrow \exists \mathbf{z} \psi(\mathbf{x}, \mathbf{z})$, where $\phi(\mathbf{x})$ (the *body*) and $\psi(\mathbf{x}, \mathbf{z})$ (the *head*) are conjunctions of atoms (referring to predicates) over a relational schema S . Atoms in TGDs' heads are referred to as *intensional (IDB atoms or IDBs for short)*, while the ones appearing only in the bodies are known as *extensional (EDB atoms or EDBs for short)*. Answers to a CQ over TGDs can be computed using the *semi-naive* algorithm [4] that adopts a bottom-up approach, starting from D and repeatedly applying the TGDs to derive new facts (possibly containing *labelled nulls* to replace existential quantification) until a fixpoint is reached. Q is answered over the materialised IDBs.

We argue that the use of logic-based Knowledge Graph Management Systems (KGMS) such as e.g. Vadalog [27] constitute a way to implement the so-coveted NeuroSymbolic (NeSy) reasoning, a subfield of Artificial Intelligence (AI) in which many researchers are conveying their efforts over the last years [146, 114, 162]. NeSy solutions promise to provide best-of-both-worlds scenarios in which the advantages of NN or ML, such as trainability from raw data and robustness against faults in the underlying data are integrated with those of symbolic approaches like formal logic, such as explainability and provable correctness.

Example 2. Consider the following Datalog rule σ :

$$\begin{aligned} & \text{img_class}(\text{img}_1, \text{pred_id}_1, \mathbf{pred_1}), \\ & \text{img_class}(\text{img}_2, \text{pred_id}_2, \mathbf{pred_2}), \\ & \text{inherited_hypernim}(\text{pred_id}_1, \mathbf{hypernim}), \\ & \text{inherited_hypernim}(\text{pred_id}_2, \mathbf{hypernim}) \\ & \rightarrow \text{related}(\text{img}_1, \text{img}_2, \mathbf{hypernim}). \end{aligned}$$

where `img_class` is a neural predicate [114] comprising predictions coming from a Convolutional Neural Network (CNN) such as EfficientNet [160], and `inherited_hypernim` is a predicate containing knowledge derived from the online lexical database WordNet [120]. Additionally, EfficientNet provides a truth value for each prediction. Combining these ingredients into a unified framework for neuro-symbolic reasoning such as MV-Datalog (introduced in Chapter 3) allows to apply the expert knowledge expressed by σ on partially true statements coming from EfficientNet and certain knowledge coming from WordNet. In Figure 1.3 it is possible to see an example of such NeSy integration.

The scenario presented in Example 2 showcases the integration of (i) expert knowledge embedded in logical rules, (ii) large available datasets and (iii) NN outputs embedded in neural predicates.

Optimising logical reasoning

In developing NeSy approaches, one must integrate one side of the medal with the other without sacrificing the benefits of either of the two. Preserving as much as possible the performances offered by the Neural approaches is of particular importance to allow for the formalism to be used in a wide range of applications. Matching the throughput offered by pre-trained NNs or ML algorithms using Symbolic approaches is inherently impossible since transforming a given input with a single pass over, e.g. a NN, would require significantly less time than performing complex reasoning tasks in, e.g. Datalog. Nevertheless, a wide range of tasks are solvable by either formal logic approaches or ML ones but not both. Therefore, no accurate comparison between the performances of the two can be made for such tasks. Trying to minimise the gap between the two to allow for a smooth integration is a constant effort in the literature (see, e.g. Section IV of [43]). Continuous theoretical and technical improvements have led to the development of highly efficient Datalog systems for widespread practical use

1. Introduction

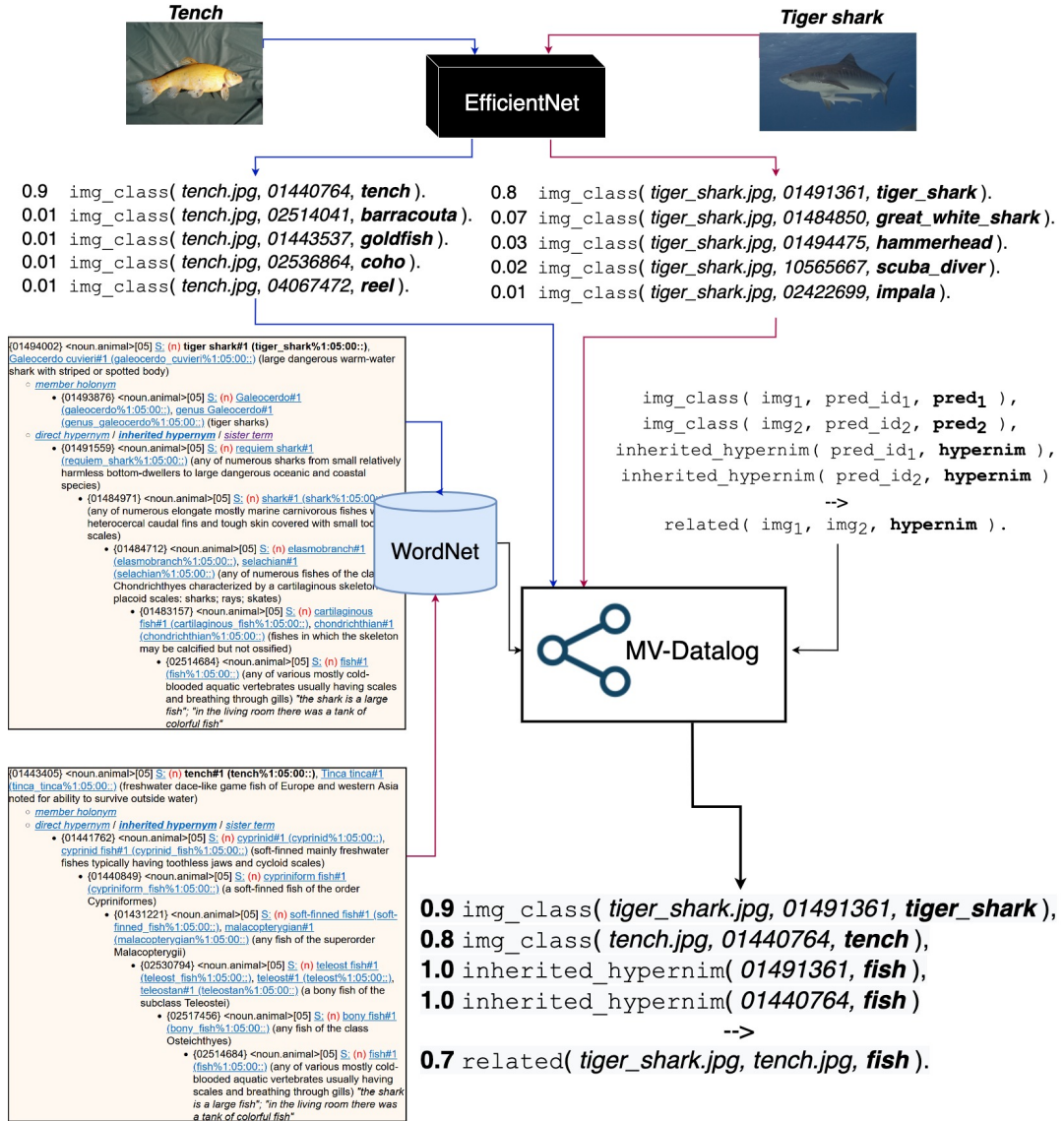


Figure 1.3: The present Figure shows the integration of a Datalog rule together with predictions coming from the neural network EfficientNet and knowledge from the lexical database WordNet. These inputs are combined by the MV-Datalog ontological language (see Chapter 3) to infer that the images *tiger_shark.jpg* and *tench.jpg* are related by the relation **fish** as this is a common hypernym. This relation has associated a truth value of **0.8**.

in various applications (see, e.g., [113]). Datalog would represent the application's bottleneck in any system integrating Datalog with any pre-trained NN or ML application. In this work, we propose a rewriting algorithm for a fragment of Datalog called Datalog[±] Piece-Wise Linear $\cup$ Warded (PWLW), allowing for a space and time-efficient computation while preserving CQ-equivalence (see [14]). After an extensive investigation, it has been discovered that more than 70% of the problems addressed in industry-oriented scenarios can be modelled via PWLW programs [31]. Recent work has shown that ontological reasoning under warded TGDs with piece-wise linear recursion is NLOGSPACE-complete in data complexity, arguing that such complexity is the best lower bound possible for a reasoning language that supports powerful graph navigation and, at the same time, non-trivial ontological query answering. Yet, the algorithm proving the membership to the NLOGSPACE class is based on a reduction to the problem of checking whether a proof tree with certain properties exists, which is far away from a practical reasoning algorithm. In Chapter 4, we exploit such results and prove that every warded set of TGDs with piece-wise linear recursion can be "linearized", i.e., transformed into a conjunctive query-equivalent set of linear TGDs, where every TGD has at most one intensional predicate in the body. We extend the results on the linearization to PWLW programs in many-valued settings. We provide and implement a general linearization technique in the Vatalog system, a state-of-the-art Datalog-based reasoner. We describe how such a system exploits the linearization procedure to enable a fully-streaming and space-efficient reasoning architecture.

Automatization of logical rules discovery

Writing logical rules is a time-consuming and expensive task as it requires employing domain experts to model the constraints representing the application's domain as much as possible. Nevertheless, logical programs can hardly fully embed all the complexities of a given domain. To reduce the efforts in writing logical rules, several methods to automatically discover logical rules have been proposed [69], amongst which Association Rules Mining (ARM) [7] is an exciting one which has grasped the attention of many researchers since the '90s.

ARM is a family of rule-based ML methods for discovering relationships between large datasets' variables. Association rules correlate the co-occurrence of attribute values to predict other attribute values from a large structured dataset. Initially developed in the context of market basket data analysis, ARM algorithms have been widely applied in several contexts for different purposes. Association rules

are of the form $FP \rightarrow head$, where FP is the so-called frequent pattern and $head$ is the predicted attribute value. From this, it is easy to see a straightforward correspondence between Datalog rules and association rules. Mining Datalog rules is fundamental to model all the constraints representing a specific domain as much as possible.

Unfortunately, not much research has been conducted in the context of mining association rules from uncertain data. Most of the research focused on the problem of handling fuzziness interpreted as vagueness of the concept [121, 140, 110]. On the other hand, we have focussed on handling partially true data points. Consider the following example:

Example 3. Let D be a database containing observations coming from a crowd of Internet of Things (IoT) sensors which upload data in a centralised table with scheme `sensor(ID, Label, Data, Precision, Timestamp)`, where `ID` is a unique identifier of the sensor, `Label` = {*temperature, humidity, wind_speed, pressure, rainfall*} contains the name of the data observed by the sensor, `Data` contains the readings of the data and `Precision` is the accuracy of the measurement. Examples of data points are:

```
sensor(1, humidity, low, 0.95, 2023-01-01_00:00:00)
sensor(2, wind_speed, medium, 0.95, 2023-01-01_00:00:00)
```

Classical methods to mine association rules from a dataset with a schema such as the one presented in Example 3 would either ignore the field `Precision` or, at best, use a threshold to distinguish the high-quality data points from the low-quality ones, ignoring the latter. Moreover, extracting association rules from data whose attributes range over a dense domain represents a further challenge. To overcome this, a small finite domain is used for each attribute. Examples of mined

association rules from Example 3 domain might include:

```
sensor(ID1, Label1, Data1, Precision1, Timestamp),  
Label1 = humidity, Data1 = high,  
sensor(ID2, Label2, Data2, Precision2, Timestamp),  
Label2 = temperature, Data2 = low  
→ weather = cloudy.
```

```
sensor(ID1, Label1, Data1, Precision1, Timestamp),  
Label1 = humidity, Data1 = low,  
sensor(ID2, Label2, Data2, Precision2, Timestamp),  
Label2 = temperature, Data2 = high  
→ weather = sunny.
```

```
sensor(ID, Label, Data, Precision, Timestamp),  
Label = wind_speed, Data = high,  
→ weather = stormy.
```

Moreover, many association rules could be produced to cover all the corner cases, depending on the nature of the data. How can these be filtered down to the relevant ones? How can a domain expert review them without being overwhelmed by the amount of information to review?

In addition, while several methods have been proposed to evaluate the quality of mined association rules, there is no unique approach to integrate automatic, semi-automatic and manual evaluations easily. For example, we propose an integrated approach to automatically and semi-automatically mine association rules in the context of fuzzy data.

Contributions

We propose extensions of Datalog and Datalog[±], i.e., Datalog with existential quantification in the head, to the many-valued semantics of infinite-valued Łukasiewicz logic. We argue that these languages offer natural semantics for practical applications with uncertain data while preserving key theoretical properties of the classical (two-valued) case. Moreover, we report on an experiment carried out in

collaboration with experts from the Central Bank of Italy involving the use of Datalog over the many-valued semantics. We present an optimization procedure for Datalog programs which allows space and time efficiency. Finally, we also present a unified methodology to mine and use association rules including methods to handle fuzzy data.

In the following, we report a schematic overview of the main contributions reported in this document:

- We introduce MV-Datalog as Datalog with Łukasiewicz semantics. We discuss adaptations of the standard minimal model concept to fuzzy models where every fact takes the least possible truth degree. We show that unique minimal models always exist for consistent inputs and give a characterisation of such minimal models. We also show that these results hold for K -fuzzy models, i.e., interpretations that satisfy formulas of the program only to a degree K rather than fully.
- We show that MV-Datalog is suitable for practical applications. For one, MV-Datalog characterisation reveals that finding K -fuzzy models is feasible by building on standard methods as from the classical setting. In particular, we also show that fuzzy fact entailment in MV-Datalog has the same complexity as fact entailment in Datalog.
- We propose MV-Datalog[±] as a practice-oriented extension of MV-Datalog by existential quantification in the head. To obtain intuitive semantics for logic programming, we deviate from the standard semantics of many valued existential quantification and instead argue for the use of strong existential quantification. We discuss fundamental model-theoretic properties and challenges of MV-Datalog[±] and show how our characterisation results for MV-Datalog can be extended to capture a natural class of preferred models of MV-Datalog[±], providing the foundation for a system implementation of MV-Datalog[±].
- We report on a practical experiment on the use of MV-Datalog in a project involving the use of big data and uncertain reasoning. In particular, the problem of family link detection over the Italian companies' knowledge graph is addressed using graph-theoretic techniques and combining the results using reasoning over fuzzy data. The utilised technique allowed us to infer a considerable number of new relations, paired with a detailed explanation of the reasoning.

-
- We prove that every Piece-Wise Linear $\cup$ Warded program can be rewritten in a linear normal form, i.e. linearized, preserving the CQ-equivalence of the original and linearized program. We show that such a normal form can be leveraged during CQ answering to achieve space and time efficiency. We introduce a set of rewriting tools for Datalog[±] and use them to provide a linearization procedure. We also prove that the linearized programs preserve the CQ-equivalence in many-valued settings. Furthermore, we compare the execution times of original and rewritten PWLW programs in many-valued settings and report on the advantages of such rewritten programs in terms of time and efficiency, which showed initial performance improvements.
 - We present a patented framework to automatically discover and apply association rules from certain and/or uncertain data with low to no efforts required from human components. This incorporates classical and novel methodologies to mine high-quality association rules capable of repairing and extending database knowledge. We show how uncertain data are treated by employing novel support and confidence metrics, among other possibilities. We use a set of statistical relevance filters to assess the quality of the rules. The framework also allows for an optional seamless integration with human experts who can perform further quality controls on the mined rules. Moreover, the framework contains default values for its parameters and mechanisms to adjust them to allow for minimal efforts from experts while preserving high flexibility. The mined association rules are also automatically applied to improve the quality of databases.
 - We report on an industry-led real-world scenario in which the proposed framework to mine association rules has been used to improve the quality of the Owler crowd-sourced database. The framework allowed us to mine high-quality association rules, which allowed us to find a high number of link predictions and data smells. These results have been validated by domain experts who proved their accuracy.

Scientific Publications

The work reported in this document led to the following scientific publications:

- Matthias Lanzinger, Stefano Sferrazza and Georg Gottlob. MV-Datalog+-: Effective rule-based reasoning with uncertain observations. *Theory and Practice of Logic Programming* 22 (5), 678-692, Best Paper Award. Personal contributions to the paper: I have contributed comprehensively to all aspects of this work.
- Matthias Lanzinger, Stefano Sferrazza, Georg Gottlob. New Perspectives for Fuzzy Datalog. *Proceedings of the 4th International Workshop on the Resurgence of Datalog in Academia and Industry (Datalog-2.0 2022)*. Personal contributions to the paper: I have contributed comprehensively to all aspects of this work.
- Marta Bernardini, Matthias Lanzinger, Rosario Laurendi, Stefano Sferrazza. Family Link Detection in Uncertain Settings with MV-Datalog+—EcoFinKG, 2022. Personal contributions to the paper: I have contributed comprehensively to all aspects of this work.
- Jinsong Guo, Aditya Jami, Markus Kröll, Lukas Schweizer, Sergey Paramonov, Eric Aichinger, Stefano Sferrazza, Mattia Scaccia, Stéphane Reissfelder, Eda Cicek, Giovanni Grasso, Georg Gottlob. When Automatic Filtering Comes to the Rescue: Pre-Computing Company Competitor Pairs in Owler. *Proceedings of the ACM on Management of Data* 1 (2), 1-23. Personal contributions to the paper: I have contributed to the conceptualization and implementation of the work, other than editorial contributions.
- Georg Gottlob, Jinsong Guo, Aditya Jami, Markus Kröll, Stéphane Reissfelder, Lukas Schweizer, Eric Aichinger, Stefano Sferrazza. CompeGe: Computing Company Competitor Pairs By Knowledge Based Inference Combined With Empirical Validation. *Proceedings of the 4th International Workshop on the Resurgence of Datalog in Academia and Industry (Datalog-2.0 2022)*. Personal contributions to the paper: I have contributed to the conceptualization and implementation of the work, other than editorial contributions.
- Stefano Sferrazza, Georg Gottlob, Giovanni Grasso, Aditya Jami, Markus Kröll, Lukas Schweizer. Association-rule based data outlier detection and record completion system and method. US Patent 11,847,105. Personal

contributions to the patent: I have contributed comprehensively to all aspects of this work.

- Matthias Lanzinger, Stefano Sferrazza, Przemysław A Wałęga, Georg Gottlob. Fuzzy Datalog³ over Arbitrary t-Norms (LPAR 2024). Personal contributions to the paper: I have contributed to the conceptualization and implementation of the work, other than editorial contributions.

Thesis Structure

The structure of this document is as follows:

- Chapter 2 provides a broad overview of the background definitions and related work.
- Chapter 3 introduces the contributions to the field of many-valued logic and its usage in real-world applications.
- Chapter 4 describes a novel technique to rewrite logical programs to achieve better space and time efficiency both in crisp and many-valued settings.
- Chapter 5 describes a comprehensive approach to managing the full life-cycle of association rules, from preparing the data, mining and reviewing the association rules to automatically applying them, both in crisp and many-valued settings.
- Chapter 6 concludes the thesis with some remarks and directions for future work.

CHAPTER 2

Background and Related Work

In this section, we will introduce all the notions required to understand the subsequent chapters fully.

2.1 Databases

Databases are collections of structured or unstructured information that can be accessed, manipulated, and updated by computer programs. Here, we introduce a short background on structured databases. Specifically, the focus will be given to databases following the relational model. These can be defined from the named or unnamed perspectives, which can be used indifferently, as they retain the same modelling power. We will refer to the former in this document, which is now briefly introduced. The notions in this section follow those in [13].

To model databases, two sets are needed: *Rel* and *Att*. These are the set of relation names and attribute names, respectively. A relation is a set of data items organized as a table. Each row (equivalent to a tuple) in the table represents an entity, which is an individual entity in the real/abstract world. Each column represents an attribute, which is a property or characteristic of an entity. For example, a relation *Customer* could have attributes such as *Name*, *Address*, *Phone*, and *Email*. Each row in the relation would then contain the values of these attributes for a specific

customer. More precisely, a database tuple is defined as $t = (A_1 : a_1, \dots, A_k : a_k)$, where $\{A_1, \dots, A_k\} \in Att$ indicating that the attribute A_i of tuple t holds value a_i , for each $i \in [1, k]$. A relation instance in the named perspective is a finite set S of database tuples of the same sort $U = \{A_1, \dots, A_k\}$, which is also called the sort of the relation instance S and denoted by $sort(S)$. By nRI (for named relational instances), we denote the set of all such relation instances.

A relation attribute can have different types, such as *text*, *number*, *date*, or *boolean*. An attribute can also have constraints, such as *primary key*, *foreign key*, *unique*, or *not null*. These constraints help ensure the data's validity and integrity in the relation. For example, a primary key is an attribute or a combination of attributes uniquely identifying each row in the relation. A foreign key is an attribute that references the primary key of another relation to establish a relationship between the two relations. A relationship is a logical connection or association between entities in different relations. For example, a relation called *Order* could have a foreign key attribute called *CustomerID*, which references the primary key attribute of the *Customer* relation. This would indicate that each order belongs to a specific customer.

In the context of Databases, a schema links attribute names (for example, the attribute names *Name*, *Address*, *Phone*, *Email* mentioned earlier) and relation names (such as *Customer* used in the previous example).

Definition 1 (Named Database Schema). A named database schema is a partial function

$$S : Rel \rightarrow P_{fin}(Att)$$

such that the set of elements in S , $Dom(S)$ (i.e. the domain of S) is finite. For $R \in Dom(S)$, the sort of R under S is the set $S(R)$. The arity of R under S , denoted $ar_S(R)$, is $|S(R)|$.

If not explicitly stated, a named database schema is also called simply schema.

It is now possible to introduce the notion of a database instance as:

Definition 2 (Database Instance). A database instance D of a named schema S is a function

$$D : Dom(S) \rightarrow nRI$$

such that $sort(D(R)) = S(R)$, for every $R \in Dom(S)$.

For the sake of simplicity, usually, schema information is provided without explicitly stating the schema name as $R[A_1, \dots, A_k]$ over $S(R) = A_1, \dots, A_k$.

2.2 Graphs and Trees

The notions in this section follow those in [13]. A graph is a pair (V, E) containing the finite sets of vertices (also referred to as nodes) V and edges E . Graphs can be directed or undirected. The former have directed edges in which the order indicates the direction of the edge. For example, the edge (u, v) indicates that there is an edge from node u to v . The latter has undirected edges in which the order of the nodes does not matter as the edges do not have any direction.

A graph $G_1 = (V_1, E_1)$ is a subgraph of a graph $G_2 = (V_2, E_2)$ if $V_1 \subseteq V_2$ and $E_1 \subseteq E_2$. Moreover, if both G_1 and G_2 are directed graphs, then G_1 is the subgraph of G_2 induced by V_1 if $E_1 = E_2 \cap (V_1 \times V_1)$. If both G_1 and G_2 are undirected graphs, then G_1 is the subgraph of G_2 induced by V_1 if $E_1 = E_2 \cap \{S \subseteq V_1 \mid |S| = 1 \text{ or } |S| = 2\}$. An undirected path of length k from u_0 to u_k in a graph $G = (V, E)$ is a non-empty sequence of nodes $n = u_0 u_1 \cdots u_k$ for $k \geq 0$, where:

- if G is a directed graph, then $(u_{i-1}, u_i) \in E$ or $(u_i, u_{i-1}) \in E$ for every $i \in [k]$, and
- if G is an undirected graph, then $\{u_{i-1}, u_i\} \in E$ for every $i \in [k]$.

A graph is connected if, for every pair of nodes $u, v \in V$, there is an undirected path from u to v . A path from u_0 to u_k in a directed graph $G = (V, E)$ of length k is a non-empty sequence of nodes $n = u_0 u_1 \cdots u_k$ for $k \geq 0$, where $(u_{i-1}, u_i) \in E$ for every $i \in [k]$. A connected directed graph $T = (V, E)$ is a tree if

- for every node u , there is at most one node v with $(v, u) \in E$, called the parent of u , and
- there is exactly one node u , called the root of T , without a parent.

The notions of subtree and induced subtree are naturally derived from the concepts of subgraph and induced subgraph. Moreover, if v is the parent of u in a tree T , then u is called a child of v , and if v does not have any children, then v is called a leaf of T . Finally, if there exists a path from v to u in T , then v is said to be an ancestor of u , and u is said to be a descendant of v .

2.3 Datalog

Datalog is a rule-based logical query and programming language whose rules (in the basic version) syntactically correspond to universally quantified function-free Horn clauses. Datalog is a language tailored for relational databases and assumes a database D , for example, an enterprise database or knowledge graph, whose tables correspond to so-called EDB predicates. Here, we recall the most important concepts of Datalog, and refer the reader to standard Database textbooks [44, 14] for further details.

Given a database D , a (plain) Datalog program Σ computes an intensional database (IDB), which consists of relational instances of the so-called IDB predicate symbols in the program. For example, suppose `parent` is a binary EDB relation where `parent( $X, Y$ )` expresses that X is a parent of Y . In this case, the following Datalog program computes the IDB relation `ancestor`, which is the transitive closure of the `parent` relation. Here, the `ancestor` IDB relation is also called an IDB predicate and the name `ancestor` is an IDB predicate symbol. When the meaning is evident from the context, a predicate symbol is sometimes referred to as "predicate".

Example 4.

$$\begin{aligned} &\text{parent}(X, Y) \rightarrow \text{ancestor}(X, Y). \\ &\text{parent}(X, Y), \text{ancestor}(Y, Z) \rightarrow \text{ancestor}(X, Z). \end{aligned}$$

Syntax

Datalog programs consist of a conjunction of rules such as the two in Example 4, which are here stated in rule-style notation. In general, Datalog does not have a standardized syntax and is often defined with a Prolog-style notation. Throughout this disclosure, we will use the rule-style notation for Datalog and its extensions.

Syntactically, a Datalog program Σ corresponds to a conjunction of its rules, and each rule corresponds to a first-order logic sentence. For example, the second rule is a syntactic equivalent of the first-order logic sentence

$$\forall X \forall Y \forall Z ((\text{parent}(X, Y) \wedge \text{ancestor}(Y, Z)) \rightarrow \text{ancestor}(X, Z)).$$

However, semantically, Datalog transcends first-order logic as, for example, the transitive closure of a relation (rather than its transitivity) is not first-order definable.

Each rule σ of a Datalog program Σ has a rule head (denoted as $head(\sigma)$) containing a single atom, and a rule body (denoted as $body(\sigma)$), which consists of a comma-separated list of atoms indicating their conjunction. An atom has a predicate symbol (also referred to as simply predicate) and arguments. Each variable that appears in a rule head must also appear in the body of the same rule.

In the most basic version of Datalog, the predicate symbol of a Datalog atom is the name of an EDB or IDB relation, and the atom's arguments are variables. In other versions, comparison atoms such as, for example, $X < Y$ are admitted in rule bodies. In some versions, arguments can also be data constants of various types, for example, numerical values. An atom whose predicate is the name of an EDB relation is an EDB-atom, and its predicate is referred to as "EDB-predicate". An atom whose predicate is not the name of any EDB relation is an IDB atom, and its predicate is referred to as "IDB-predicate". A Datalog program computes the IDB from the input EDB. The IDB contains a data relation (an IDB relation) for each IDB predicate. In the context of Datalog, one identifies a tuple of a relation with a ground atom (also "ground fact"). A ground atom is an atom whose arguments are all data values (not variables). For example, if *ancestor* is a relation containing the tuple (*adam*, *delila*), then this tuple corresponds to the ground atom *ancestor(adam, delila)*. On the other hand, a set of ground atoms can always be seen as a set of relations, one for each predicate name. The arity of an atom is the number of arguments it contains. If the same predicate name occurs in a Datalog program in atoms with different arities, then this name corresponds to a separate relation for each of these arities.

In the context of Datalog, Herbrand interpretations are sets of ground atoms whose predicate symbols are EDB or IDB predicate symbols, and whose arguments range over the finite set of data elements in the EDB and the program. This set of elements is also called the active domain or the Herbrand universe $HU(D, \Sigma)$ of D and Σ . Atoms whose predicate is an EDB or IDB predicate of a Datalog program Σ are called ground atoms (w.r.t. D and Σ), and the set of all such ground atoms is referred to as the Herbrand Base $HB(D, \Sigma)$ of D and Σ .

Semantics

The standard semantics of Datalog singles out a unique "designated" (also "preferred") Herbrand interpretation for a given program Σ with a given EDB D . This designated interpretation, which is a superset of D , is the unique minimal model of $\Sigma^* \wedge D^*$ where Σ^* denotes the syntactical logical equivalent of a Datalog program Σ , as explained above, and where D^* is the conjunction of all ground atoms

in D . It is well-known that this designated interpretation can be formulated in second-order logic and fixed-point logic and can be obtained via a least-fixed-point computation that applies the rules of Σ to D iteratively, starting from D and an empty IDB, and successively deriving further ground atoms, i.e., atoms of atoms from $HB(D, \Sigma)$, and adding them to the IDB until no further ground atom can be derived. Suppose the program Σ is considered a monotonic transformation that adds zero or more facts to an interpretation. In that case, D plus all new atoms added is indeed the least fixed point of this transformation. A rule σ of Σ may be applied via a ground substitution θ , that is, a substitution that uniformly replaces the variables in $body(\sigma)$ by elements from $HU(D, \Sigma)$. A rule $\sigma \in \Sigma$ is applicable via θ if for each atom $A \in body(\sigma)$, the result $\theta(A)$ of applying θ to A is either in D or has already been derived through a previous rule application. A rule σ is unsatisfied if it is applicable, but $\theta(head(\sigma))$ is neither in D nor has been derived in previous steps, and where $\theta(head(\sigma))$ is not in the intermediate IDB build so far. Such an unsatisfied rule is "applied" (or "fired") with θ by adding the new ground atom $\theta(head(\sigma))$ to the IDB computed so far. The final result IDB for program Σ and database D can be computed by repeatedly and exhaustively identifying rules σ that are unsatisfied with some ground substitution θ , and then applying σ with θ . It is well-known that $IDB(D, \Sigma)$ computed by this process is uniquely defined and independent of the order of rule application and that $D \cup IDB(D, \Sigma)$ is precisely the above-mentioned unique designated Herbrand interpretation.

An equivalent way of defining the semantics of Datalog is via first-order entailment (well-known not to be first-order definable). Let p be an EDB predicate and D an EDB. Then a ground atom γ of the form $p(t_1, \dots, t_n)$ is in the designated model of Σ and D iff $\Sigma^* \wedge D^*$ logically entails γ (written as $\Sigma^*, D^* \models \gamma$).

In basic versions of Datalog, the EDB is read-only, which means that EDB-predicates are not allowed to appear in rule heads. However, versions of Datalog have been defined, where the EDB itself can be augmented or updated. For example, to express description logics, versions of (possibly extended) Datalog are used to infer new EDB ground facts and thus insert new tuples into the EDB.

In database terminology, a Datalog program Σ is a (possibly recursive) view-defining query that associates $IDB(D, \Sigma)$ to each given EDB D , and where $IDB(D, \Sigma)$ constitutes a view of the EDB.

Datalog Extensions

Datalog has been extended by various features, some of which are discussed below.

One important feature is negation in rule bodies. Here, a safety condition is usually imposed: each variable of a negated atom must also occur in a positive atom of the same rule body. There are various types of negation with different semantics. Stratified negation is the best-known and most-used type of negation in the database context.

The Dependency graph $DG(\Sigma)$ of a (possibly extended) Datalog program Σ is the directed graph whose vertices are the predicate symbols occurring in Σ (annotated with their arity in case the same predicate symbol occurs with multiple arities) and whose arcs are defined as follows. There is an arc between p and q if p is the predicate of the head atom of some clause of Σ , and q is the predicate of some atom occurring in the body of the same clause. Moreover, the arc between p and q is labeled by an asterisk "*", if q is the predicate of a negated atom in a rule's body whose head predicate is p . A predicate p of a Datalog program Σ depends (positively) on a predicate q if there is a path from p to q in $DG(\Sigma)$. A predicate p of a Datalog program Σ depends negatively on a predicate q if a path from p to q in $DG(\Sigma)$ has at least one labeled edge.

A predicate p of Σ is recursive if p appears on a cycle in $DG(\Sigma)$, otherwise, p is non-recursive. A (possibly extended) Datalog program Σ is stratified if no cycle of $DG(\Sigma)$ contains a labelled edge. A stratified program can be partitioned into a linearly ordered set of strata, where, starting from stratum 1 each further stratum is assigned a consecutive number. For each IDB predicate p , the rules whose head predicate is p are all in the same stratum, which is referred to as the "stratum of p " and here denoted by $stratum(p)$, and if p depends on q , then $stratum(p) \geq stratum(q)$, and if p depends negatively on q , then $stratum(p) > stratum(q)$. A stratification of a (possibly extended) stratified Datalog program can be obtained, for example, as follows. Collect into stratum 1 each rule whose head predicates do not negatively depend on any IDB predicate of Σ ; moreover, if stratum i has been defined, and there are rules left, collect into stratum $i + 1$ all rules whose head predicates depend only on predicates belonging to strata $1 \dots i$. The IDB defined by a stratified program Σ over a database D can then be computed "bottom-up" by applying the rules of Σ stratum by stratum from the lowest to the highest stratum. With each stratum i , an IDB IDB_i is computed, where, for $i \geq j$, $IDB_i \supseteq IDB_j$. Assume Σ has k strata $\Sigma_1, \dots, \Sigma_k$. Let IDB_0 designate the EDB D . A rule σ of a stratified Datalog

program Σ is applicable with a ground substitution θ to a Herbrand interpretation I if (1) for each positive (un-negated) atom A in $body(\sigma)$, $\theta(A) \in I$ and (2) for each negative literal "not A " in $body(\sigma)$, $\theta(A)$ is not in I . Moreover, σ is unsatisfied with θ if σ with θ is applicable to I , but $\theta(head(\sigma)) \notin I$. The "application" or "firing" of an applicable unsatisfied rule σ with a ground substitution θ on a Herbrand interpretation I adds the ground atom $\theta(head(\sigma))$ to I . For each i with $1 \leq i \leq k$, IDB_i is computed from IDB_{i-1} as follows. Starting with IDB_{i-1} , construct IDB_i by repeatedly and exhaustively identifying rules σ of Σ_i that are unsatisfied with some ground substitution θ , and then apply σ with θ . Note that for a negation-free (that is, plain) Datalog program Σ , this procedure precisely computes $D \cup IDB(D, \Sigma)$. For programs with negations in rule bodies, the procedure computes the single "designated" (or "preferred") Herbrand interpretation $D \cup IDB(D, \Sigma)$, which is a model of $\Sigma^* \wedge D^*$, where Σ^* defined as for plain Datalog, except that, in addition, "not" is translated into the logical negation " $\neg$ ".

Linearity in the context of Datalog is an overloaded term. The linearity that is commonly referred to in plain Datalog usually refers to programs restricted to contain at most one atom in the body. Sometimes, the term linear Datalog is used to refer to Datalog programs which contain at most one intensional atom in the body. Since this last definition was used, amongst others (e.g. see [145, 53]), by Afrati et al in [6], to distinguish from the classic interpretation of linear Datalog, we refer to this interpretation of the term "linear" as Afrati-linear. Yet another restriction based on a notion of linearity is the one used in piece-wise linear Datalog. This is defined as follows.

Definition 3 (Piece-Wise Linear Datalog [6]). A piece-wise linear (PWL) program is a Datalog program Σ such that for every clause in Σ there is at most one atom in the body whose predicate is "mutually recursive" with the predicate in the head. Two predicates p and q are said to be mutually recursive iff p "depends on" q and q "depends on" p , where "depends on" is the least relation such that: a predicate p depends on a predicate q iff there is a clause with p in its head and either q or a predicate r which depends on q in its body.

Noticeably, every linear program is also Afrati-linear, which in turn is always PWL, while these statements do not hold in the opposite direction.

2.4 Existential Datalog

The term “Existential Datalog” has been overloaded with multiple meanings in the literature. In the present disclosure, it means Datalog extended by the possibility of using existential rules. An existential rule is a rule having existentially quantified variables in its head. By this definition, Existential Datalog (also referred to as Datalog³) [138] generalizes plain Datalog. This has been introduced to overcome the limited expressivity of plain Datalog as discussed in [131]. Note that in the context of database theory, existential rules are often referred to as “tuple-generating dependencies (TGDs)”. Commonly, the family of Datalog-based languages that include the use of variables existentially quantified in the head of rules is also referred to as Datalog[±], where the “+” refers to the extension of Datalog to new features, such as the use of existentially quantified variables, and the “-” refers to restrictions applied to the language used to achieve tractability or decidability.

It is possible to define TGDs through the notion of homomorphism.

Definition 4 (Homomorphism [37]). A homomorphism from a set of atoms A_1 to another set of atoms A_2 is a mapping h from $dom(A_1)$ to $dom(A_2)$ such that the following conditions hold: (i) if c is a constant, then $h(c) = c$; (ii) $h(A_1) \subseteq A_2$, i.e., if an atom a is in A_1 , then the atom $h(a)$ is in A_2 . In this case, it is said that A_1 maps to A_2 via h .

Building on this, TGDs can be defined as follows.

Definition 5 (Tuple-Generating Dependencies [37]). Given a relational schema S , a TGD σ over S is a first-order formula of the form $\forall X \forall Y \Phi(X, Y) \rightarrow \exists Z \Psi(X, Z)$. Such a dependency is satisfied in an instance B of S if, whenever there is a homomorphism h that maps the atoms of $\Phi(X, Y)$ to atoms of B , there exists an extension h_2 of h (i.e., $h_2 \subseteq h$) that maps the atoms of $\Psi(X, Y)$ to atoms of B .

The Herbrand Universe $HU(D, \Sigma)$ and Herbrand Base $HB(D, \Sigma)$ for a database D and an Existential Datalog program Σ are defined in exactly the same way as for plain Datalog. In the same way as plain Datalog, Existential Datalog can be extended by stratified negation. The notions of dependency graph $DG(\Sigma)$ and stratified negation carry over from Datalog to the more general language of Existential Datalog.

While plain Datalog (even when extended by stratified negation) cannot express certain ontological reasoning tasks, such tasks can be expressed in Existential

Datalog. For example, the DL-Lite description logic axiom $house \sqcap \neg isolated \sqsubseteq \exists neighbor_property$ which expresses that if a house is not isolated, there must be a neighbouring property to this. This can be rephrased in Existential Datalog (here also using stratified negation) as:

$$house(X), \neg isolated(X) \rightarrow neighbor_property(X, Y).$$

In this example, *house* and *isolated* are EDB relations. From these, the IDB relation *neighbor_property* is created. The Y variable in the rule head is to be understood as existentially quantified. Generally, all variables that appear in the head but not in the body of a rule are to be considered existentially quantified.

The logical formula syntactically corresponding to an existential rule is an implication as above, where all body variables are universally quantified and all existential variables, i.e., those appearing in the head but not in the body, are existentially quantified. The above rule thus corresponds to the logical sentence:

$$\forall x ((house(x) \wedge \neg isolated(x)) \rightarrow \exists y neighbor_property(x, y)).$$

The logical formula Σ^* , syntactically corresponding to an existential (and possibly stratified) Datalog program Σ , is the conjunction of the logical correspondents of its clauses.

For Existential Datalog, three relevant problems are considered: (1) the query-of-tuple (QOT) problem, (2) the conjunctive query (CQ) answering problem, which generalizes the QOT-problem, and (3) the certain-tuple computation problem. These problems are logically defined as follows

Definition 6 (Query-Of-Tuple (QOT) problem). An instance (D, Σ, γ) of the QOT problem consists of an EDB D , an Existential Datalog program Σ and a ground atom $\gamma \in HU(D, \Sigma)$, sometimes also referred to as "goal". The problem is to determine whether γ is a logical consequence of $D^* \wedge \Sigma^*$.

Definition 7 (Conjunctive Query Answering (CQ-answering or CQA) problem). An instance (D, Σ, Q) of the CQ-answering problem consists of an EDB D , an Existential Datalog program Σ and a conjunctive query (CQ) Q . A CQ Q is a conjunction $\exists x_1 \dots \exists x_r \Phi(x_1, \dots, x_r, y_1, \dots, y_s)$ of atoms over variables $x_1, \dots, x_r, y_1, \dots, y_s$ where $x_1, \dots, x_r$ are existentially quantified, and where $y_1, \dots, y_s$ are free. A "certain answer" to Q (relative to D and Σ) is a tuple $(a_1, \dots, a_s)$ of elements from $HU(D, \Sigma)$ such that $D^* \wedge \Sigma^*$ logically entails the closed formula $\exists x_1, \dots, \exists x_r \Phi(x_1, \dots, x_r, t_1, \dots, t_s)$ resulting from Q by replacing y_i by a_i for $1 \leq i \leq s$. The CQ-answering problem for

an instance (D, Σ, Q) is to determine all certain answers to Q relative to D and Σ . A variant of this problem is to determine whether a certain answer exists, and if so, to find one certain answer. A special case of a CQ is a boolean conjunctive query (BCQ). This is a CQ all of whose variables are existentially quantified. Such a BCQ Q is answered "true" if $D^* \wedge \Sigma^*$ logically entails Q . Otherwise it is answered "false".

Definition 8 (Certain Tuples Computation problem). An Instance (D, Σ) of the Certain Tuple Computation Problem consists of an EDB D and an Existential Datalog program Σ . The problem is then to compute all certain atoms for (D, Σ) , that is, all atoms $R(a_1, \dots, a_r)$ entailed by $D^* \wedge \Sigma^*$ where R is a predicate symbol in D or Σ of arity r , and where for $1 \leq i \leq r$, $a_i \in HU(D, \Sigma)$.

Unfortunately, the above-mentioned problems are undecidable for Datalog[∃] programs [37]. Therefore, various syntactic restrictions on Existential Datalog programs have been considered for which these problems are decidable. Some of these are discussed further below. The decision procedures for most decidable Existential Datalog fragments use a procedure referred to as "Chase", which will be described next.

The Chase

Given that various different types of Chase are in the literature, a clarification is needed. Here, by "Chase" we refer to the "oblivious Chase" which has been thoroughly described in the literature [111, 112, 60, 4, 24, 9].

The Chase is for Existential Datalog what the above-described least-fixed-point computation is for plain Datalog. In fact, the Chase of an Existential Datalog program Σ applied to a database D applies unsatisfied rules of Σ over and over and stops only if no unsatisfied rule is left. However, there are a number of important aspects in which the Chase differs from the already described least-fixed-point iteration procedure for plain Datalog.

First, the applicability of a rule (called oblivious applicability) at a stage in the Chase process is defined similarly as for plain Datalog, except that the ground substitutions θ in consideration now replace the variables in the body of a rule by elements of $HU(I, \Sigma)$, rather than just $HU(D, \Sigma)$, where I is the set (interpretation) containing all atoms of D plus those previously derived. This is because (as will be explained below) existential rules can create new elements that do not appear in D nor in Σ .

2. Background and Related Work

Second, at any given stage in the Chase, an interpretation I does not satisfy a rule σ with a ground substitution θ from the body variables of σ to elements of $HU(I, \Sigma)$ if (1) σ has not yet been applied with θ and (2) for each atom $A \in \text{body}(\sigma)$, $\theta(A) \in I$. One then says that σ with θ is unsatisfied by I (or unsatisfied in I).

Third, the application of a rule σ of an Existential Datalog program Σ with a ground substitution θ to an interpretation I (containing the EDB D) by the Chase needs to take care of existentially quantified variables, that is, of all those variables that appear in the head of σ but not in the body of σ . Assume the body variables of σ are $X_1, \dots, X_r$, and the existentially quantified variables in the rule head are $Y_1, \dots, Y_s$. Then θ^+ is the extension of θ defined as follows: for $1 \leq i \leq r$, $\theta^+(X_i) = \theta(X_i)$ and for $1 \leq j \leq s$, $\theta^+(Y_j) = v_j$, where v_j is a fresh null value (technically referred to as Skolem constant), i.e., a fresh artificial data element that is recognizable as such and does not occur in $HU(I, \Sigma)$. The application of σ with θ to a given interpretation I then generates I^+ by adding the new ground atom $\theta^+(\text{head}(\sigma))$, that is, the atom resulting from the head atom of σ by replacing each variable Z in this head atom by $\theta^+(Z)$. The newly created null values $HU(I^+, \Sigma)$ become members of the Herbrand universe $HU(I^+, \Sigma)$ which is defined as $HU(I^+, \Sigma) = HU(I, \Sigma) \cup \{v_1, \dots, v_s\}$.

Fourth, note that the chase may generate infinite new atoms by the above definition.

Example 5. Let Σ_{par} be the program containing the following two rules:

$$\begin{aligned} \text{person}(X) &\rightarrow \text{parent}(X, Y). \\ \text{parent}(X, Y) &\rightarrow \text{person}(Y). \end{aligned}$$

When starting with an EDB D_0 containing the unique atom $\text{person}(\text{john})$ the result of the Chase would be the infinite set of atoms $\{\text{person}(\text{john}), \text{parent}(\text{john}, c_1), \text{person}(c_1), \text{parent}(c_1, c_2), \text{person}(c_2), \text{parent}(c_2, c_3), \dots\}$, where each C_i is a new Skolem constant. The idea that the chase may have an infinite result is intended and will be discussed in more detail further below. However, with recursive existential rules, there is a pitfall to avoid: a group of rules may be used infinitely often over and over and may preempt other rules from firing. For instance, let Σ'_{par} be the program obtained from Σ_{par} by adding the rule

$$\text{parent}(X, Y) \rightarrow \text{child}(Y, X).$$

If one applies the rules in the wrong order, this rule may never fire. In fact, if the Chase is allowed to fire the rules in any order, then a particular execution

sequence of rule applications by the Chase may give priority to the rules of Σ_{par} , thereby infinitely delaying the application of the child rule. Thus, no child atom would ever be generated, and the result of all facts produced would be the same as the above infinite set obtained by chasing Σ_{par} over D_0 . To avoid such unintended behaviour, one requires the Chase to be fair, which means that any atom that can be at all derived from the EDB by applications of unsatisfied rules (applied in whatever order) must be derivable by the Chase after a finite number of rule applications. There is still a lot of freedom in using different rule application orderings when implementing a fair Chase algorithm. However, it is known that fairness guarantees the uniqueness of the Chase (up to the names of Skolem constants). Let $chase_1$ and $chase_2$ be two realizations of the fair Chase that possibly use different orders of rule application, and let $chase_1(D, \Sigma)$ and $chase_2(D, \Sigma)$ be the sets of all atoms generated when applying Σ over database D via $chase_1$ and $chase_2$, respectively. Then $chase_1(D, \Sigma)$ and $chase_2(D, \Sigma)$ are isomorphic and differ only in the names of the null values (i.e. Skolem constants) generated by $chase_1$ and $chase_2$. The names of the null values are completely irrelevant to the QOT Problem, the CQA Problem, and the Certain Tuples Computation Problem, as well as to most other relevant problems. Therefore, one safely writes $Chase(D, \Sigma)$ to express the result of applying any algorithmic realization of the fair Chase, keeping in mind that applying any other fair Chase realization to D and Σ would yield an isomorphic result.

It is well-known that $Chase(D, \Sigma)$ is extremely useful for answering the QOT problem, the CQ-answering problem, and the certain-tuple computation problem. In particular, the following holds:

- an instance (D, Σ, γ) of the QOT problem is a yes-instance (i.e., a positive instance), if and only if $\gamma \in Chase(D, \Sigma)$;
- the solution to an instance (D, Σ, Q) of the CQ-answering problem, where Q is of the form $\exists x_1 \dots \exists x_r \Phi(x_1, \dots, x_r, y_1, \dots, y_s)$ as above, is precisely the set of all ground tuples $\gamma = (a_1, \dots, a_s)$, such that for $1 \leq i \leq s$, $a_i \in HU(D, \Sigma)$ and γ belongs to the answer of Q over $Chase(D, \Sigma)$;
- for an instance (D, Σ) of the Certain-Tuple Computation Problem, the solution precisely consists of the set of all ground tuples of $Chase(D, \Sigma)$.

From this, it follows that if an implemented Chase procedure to compute $Chase(D, \Sigma)$ terminates normally, i.e., does not terminate abnormally due to a time limit, memory overflow, or similar reasons, then $Chase(D, \Sigma)$ can be used effectively for the

algorithmic solution of the three mentioned problems. In addition, (D, Σ) can be viewed as a knowledge graph with fact-based (extensional) and rule-based (intensional) knowledge, and thus, in case an implemented Chase procedure to compute $\text{Chase}(D, \Sigma)$ terminates normally, the Chase procedure computes a materialization of the knowledge graph (D, Σ) , that can be stored in a classical database and used, as explained above, for the effective algorithmic solution of essential algorithmic decision and computation problems.

A very useful rule application strategy to compute $\text{Chase}(D, \Sigma)$ is level saturation. This strategy organizes the rule applications into levels, and proceeds level by level, starting with interpretation $I_0 = D$ and successively generating interpretation I_{i+1} from interpretation I_i as long as possible, by exercising all rule applications belonging to level i , which are precisely all applications of all rules σ with substitution θ that are unsatisfied in I_i . Thus, at the beginning all those pairs (σ, θ) are identified as unsatisfied in the EDB. Then, these are applied and the generated atoms are added to $I_0 = D$. The same process is applied to I_1 , and so forth. This guarantees that any atom that is derivable from D by the rules of Σ will eventually be derived, and thus guarantees fairness.

The level-saturating chase can be used to define the derivation depth $ddepth(A, D, \Sigma)$ of each atom A in $\text{Chase}(D, \Sigma)$: $ddepth(A, D, \Sigma)$ (or just $ddepth(A)$ if D and Σ are understood) is the smallest i such that $A \in I_i$, where $i \in \mathbb{N}$.

If ϕ is a ground atom or a Boolean conjunctive query, then ϕ is a logical consequence of $D^* \wedge \Sigma^*$, denoted by $D^* \wedge \Sigma^* \models \phi$, if and only ϕ is satisfied in (and thus a logical consequence of) $\text{Chase}(D, \Sigma)$, denoted by $\text{Chase}(D, \Sigma) \models \phi$.

Dealing with infinite Chase results

Various approaches deal with the possible infinity of the set of atoms $\text{Chase}(D, \Sigma)$.

One approach is to leave it to the Existential Datalog programmer to ensure that the Chase will eventually stop, and thus generate a finite result $\text{Chase}(D, \Sigma)$. This is akin to normal programming, where the programmer is responsible for ensuring program termination.

Another approach is to compute $\text{Chase}(D, \Sigma)$ only partially and stop the derivation of further atoms (i) after a certain number of atoms have been derived and/or (ii) a certain number of null values (Skolem constants) have been generated, or a certain derivation depth has been attained. These "cut-off" numbers in (i)-(iii) could either be fixed or be given in the input, or determined by some chosen

computation method M via a function $f_M(D, \Sigma, \text{maxqsiz})$, that depends on the input database D (usually an EDB) and/or on the program Σ , and/or on a maximum query size maxqsiz . Here, the maximum query size is the maximum number of atoms of a conjunctive query (or union of conjunctive queries) that one expects to be issued against the computed database $D \cup \text{IDB}$.

Other approaches look for syntactic restrictions of Existential Datalog that guarantee that the Chase terminates after a finite number of steps. The best known amongst these restrictions is *weak acyclicity* [61].

Definition 9 (Weakly Acyclic programs). Let Σ be a set of TGDs over a schema S . To check whether Σ is weakly acyclic, one constructs a position dependency graph $PDG(\Sigma)$ whose vertices are predicate argument positions (rather than predicates). For example, an n -ary predicate p has n positions $p[1], \dots, p[n]$. There is an arc $p[i] \rightarrow q[j]$ from $p[i]$ to $q[j]$ if there is a rule σ in Σ with head-predicate q , and having a p -atom A in the body such that (i) position $p[i]$ of A contains a variable that also occurs in position $q[j]$ in the head of σ , or (ii) an existentially quantified variable occurs in position $q[j]$ of the head of σ . Moreover, if case (ii) applies, the arc is labelled by a special symbol (e.g. by " $\exists$ "), which is here denoted by $p[i] \xrightarrow{\exists} q[j]$. A program Σ is weakly acyclic if no labelled arc occurs on any cycle of $PDG(\Sigma)$.

Weakly acyclic programs Σ have two important properties: (i) $\text{Chase}(D, \Sigma)$ is finite for each database D , and (ii) the Chase (where rules may be applied in any order) stops automatically. An example of a weakly acyclic program is the following program Σ_{LTC} (for "labelled transitive closure"), whose only EDB relation is the edge and where tc and labelled are IDB relations:

$$\begin{aligned} \text{edge}(X, Y) &\rightarrow \text{tc}(X, Y). \\ \text{edge}(X, Y), \text{tc}(Y, Z) &\rightarrow \text{tc}(X, Z). \\ \text{tc}(X, Y) &\rightarrow \text{l tc}(X, Y, Z). \end{aligned}$$

The position dependency graph $PDG(\Sigma_{LTC})$ has as vertices all argument positions of each of its three predicates, and has the following arcs:

$$\begin{aligned} \text{edge}[1] &\rightarrow \text{tc}[1], \text{edge}[2] \rightarrow \text{tc}[2], \text{tc}[2] \rightarrow \text{tc}[2], \text{tc}[1] \rightarrow \text{l tc}[1], \\ \text{tc}[2] &\rightarrow \text{l tc}[2], \text{tc}[1] \xrightarrow{\exists} \text{l tc}[3], \text{tc}[2] \xrightarrow{\exists} \text{l tc}[3]. \end{aligned}$$

While there is a cycle in this graph, namely the loop $tc[2] \rightarrow tc[2]$, there is no cycle passing through a labelled arc, hence Σ_{LTC} is weakly acyclic. It has been proved that CQ-answering is PTIME-complete in data complexity [61, 55] and 2EXPTIME-complete in combined complexity [61, 40].

Generalizations of weak acyclicity which enjoy the same properties (i) and (ii) are super-weak acyclicity [115] and joint acyclicity [96].

More sophisticated syntactic restrictions have been studied that, even though $Chase(D, \Sigma)$ may be infinite, guarantee that queries up to a given size $maxqs$ can be answered within a finite initial fragment of the level-saturating chase. Since the QOT problem and the Certain Tuple Computation problem can clearly be reduced to queries having $maxqs = 1$, also these two latter problems are solvable within this finite initial fragment. Therefore, the three essential problems become decidable under each of these syntactic restrictions.

A first example includes the so-called linear Existential programs [76].

Definition 10. Let Σ be a set of TGDs over a schema S . Σ is a linear existential program iff for each rule $\sigma \in \Sigma$, $|body(\sigma)| = 1$.

Note that, in case of stratified negation, the atom in $body(\sigma)$ must be positive. An example of a linear Existential Datalog program is the program in Example 5. As already explained, even for databases D with just a single person atom, $Chase(D, \Sigma_{par})$ is infinite. For a conjunctive query Q , let $sz(Q)$ denote its size, which is here the number of atoms that are part of it. There exists a function $f_{lin}(\Sigma)$ such that for each linear Existential Datalog program Σ , for every database D and conjunctive query Q , the answer to Q over $Chase(D, \Sigma)$ is equal to the answer to Q over the first $sz(Q) \times f_{lin}(\Sigma)$ levels of $Chase(D, \Sigma)$, which is a computable finite set. Concretely, this has been shown for $f_{lin}(\Sigma) = \#pred \times (2\omega)^\omega$, where $\#pred$ is the number of predicate symbols of Σ , and ω is the maximum of the arities of the predicates of Σ .

A more general class is the class of frontier-guarded Existential Datalog programs [78].

Definition 11 (Frontier-Guarded Existential Datalog). We say that a rule σ is frontier-guarded if there exists an atom $A \in body(\sigma)$, called the frontier-guard and denoted by $fg(\sigma)$, such that A contains all the variables appearing both in $body(\sigma)$ and $head(\sigma)$ (i.e. non-existentially quantified variables). A program Σ is frontier-guarded if all its rules are frontier-guarded.

A rule is called "guarded" if its frontier-guard contains all the variables appearing in all the other atoms in the rule body; in this case, frontier-guard is simply called "guard".

Definition 12 (Guarded Existential Datalog). An Existential Datalog program Σ is called guarded if its frontier-guard contains all the variables appearing in all the other atoms in the rule body; in this case, frontier-guard is simply called "guard".

It has been proven that guarded existential Datalog is PTIME-complete in data complexity [38] and 2EXPTIME-complete in combined complexity [37].

Definition 13 (Guarded Chase Forest). Let D be a database and Σ a frontier-guarded Existential Datalog program, then, the guarded Chase forest $GCF(D, \Sigma)$ is a forest (i.e, a multi-rooted acyclic directed graph) whose vertices are labelled by the atoms from $Chase(D, \Sigma)$, (each atom from $Chase(D, \Sigma)$ labels at least one vertex but possibly multiple vertices), which is obtained from D and Σ as follows. Each atom A in D gives rise to a root node of $GCF(D, \Sigma)$. If in the construction of $Chase(D, \Sigma)$, a rule σ having frontier-guard G is applied and generates an atom $B = \theta(head(\sigma))$ via a substitution θ , then each node labelled $\theta(G)$ has a child labelled B . For each atom C , the guarded depth $gdepth(C, D, \Sigma)$ of an atom C in $Chase(D, \Sigma)$ is zero for each root node of $GCF(D, \Sigma)$ and otherwise is the length of any shortest path from a root to a vertex labelled C in $GCF(D, \Sigma)$. Notice that, in general, an atom's guarded chase depth differs from its derivation depth (but for linear Existential Datalog programs, these notions coincide). For each positive integer i denote by $Chase_g^i(D, \Sigma)$ be the set of all atoms A in $Chase(D, \Sigma)$ such that $gdepth(A, D, \Sigma) \leq i$.

There exists a function f_g such that for each frontier-guarded (and thus also for each guarded) Existential Datalog program Σ and every database D and conjunctive query Q , the answer of Q over the possibly infinite set $Chase(D, \Sigma)$ is equal to the answer to Q over the finite interpretation (i.e., atoms set) $Chase_g^{sz(Q) \times f_g(\Sigma)}(D, \Sigma)$. Concretely, this has been shown for the double-exponential function $f_g(\Sigma) = \#pred \times (2\omega)^\omega \times 2^{\#pred \times (2\omega)^\omega}$, where $\#pred$ and ω are, as above, the number of predicate symbols of Σ and the maximum of the arities of the predicates of Σ , respectively.

The classes of linear, guarded, and frontier-guarded Existential Datalog programs do not generalize plain Datalog. For example, the transitivity rule

$$r(X, Y), r(Y, Z) \rightarrow r(X, Z)$$

is not frontier-guarded (the frontier variables X and Z are not jointly covered by any body atom) and therefore not guarded, let alone linear. For this reason, classes of Existential Datalog programs that generalize guarded and or frontier-guarded Existential Datalog and at the same time generalize plain Datalog have been defined. The principle of these classes is that body variables that occur in special positions, called unaffected predicate positions, do not have to be "guarded", that is, do not have to be covered by a guard or frontier-guard. A predicate position of an Existential Datalog program Σ is unaffected if it is not affected.

Definition 14 (Affected Positions [37]). Given a relational schema S and a set of TGDs Σ over S , a predicate position $p[i]$ of a predicate p of S is affected with respect to Σ if either

- for some $\sigma \in \Sigma$, an existentially quantified variable appears in $p[i]$ in $head(\sigma)$,
- for some $\sigma \in \Sigma$, the variable appearing at position $p[i]$ in $head(\sigma)$ also appears in $body(\sigma)$, and only at affected positions.

We call non-affected positions all the predicate positions which are not affected.

For whatever database D , no null value (Skolem constant) will ever occur in an unaffected predicate position of any atom in $Chase(D, \Sigma)$. This means that only finitely many elements occur in unaffected positions. A weak guard of an Existential Datalog rule σ is an atom in the body of σ covering all those body variables of σ that do not appear in any unaffected predicate position in the body of σ . A weak frontier-guard of an Existential Datalog rule σ is an atom in the body of σ covering all those frontier variables of σ that do not appear in any unaffected predicate position in the body of σ . A rule σ of an Existential Datalog program Σ is weakly (frontier) guarded if it has a weak (frontier) guard.

Definition 15 (Weakly (Frontier-)Guarded Datalog). A program Σ is weakly (frontier) guarded [78] if each of its rules is weakly (frontier) guarded.

In analogy to the chase forest for guarded programs, the weakly guarded chase forest $WGCF(D, \Sigma)$ for a database D and a weakly guarded Existential Datalog program Σ is a forest (i.e, a multi-rooted acyclic directed graph) whose vertices are labelled by the atoms from $Chase(D, \Sigma)$, (each atom from $Chase(D, \Sigma)$ labels at least one vertex but possibly multiple vertices), which is obtained from D and Σ as follows. Each atom A in D gives rise to a root node in $GCF(D, \Sigma)$. If in the

construction of $Chase(D, \Sigma)$, a rule σ having weak frontier-guard G is applied and generates an atom $B = \theta(head(\sigma))$ via a substitution θ , then each node labelled $\theta(G)$ has a child labelled B . For each atom C , the weakly guarded depth $wgdepth(C, D, \Sigma)$ of an atom C in $Chase(D, \Sigma)$ is zero for each root node of $WGCF(D, \Sigma)$ and otherwise is the length of any shortest path from a root to a vertex labelled C in $WGCF(D, \Sigma)$. For each positive integer i denote by $Chase_{wg}^i(D, \Sigma)$ be the set of all atoms A in $Chase(D, \Sigma)$ such that $wgdepth(A, D, \Sigma) \leq i$.

There exists a function $f_{wg}(D, \Sigma)$ such that for each weakly frontier-guarded (and thus also for each weakly guarded) Existential Datalog program Σ and every database D and conjunctive query Q , the answer to Q over the possibly infinite set $Chase(D, \Sigma)$ is equal to the answer to Q over the finite interpretation (i.e., atoms set) $Chase_{wg}^{sz(Q) \times f_{wg}(D, \Sigma)}(D, \Sigma)$. Concretely, this has been shown for the double-exponential function

$$f_{wg}(D, \Sigma) = \#pred \times (|dom(D)| + \omega)^\omega \times 2^{\#pred \times (|dom(D)| + \omega)^\omega}$$

where $\#pred$ and ω are, as before, the number of predicate symbols of Σ and the maximum of the arities of the predicates of Σ , and where $dom(D)$ designates the active domain of the database D , that is, the data elements occurring as arguments in atoms of D , and $|dom(D)|$ denotes the cardinality (i.e., the number of elements) of the set $dom(D)$. It has been proven that both weakly guarded and weakly frontier guarded existential Datalog are EXPTIME-complete in data complexity [19], thus intractable.

A relevant subclass of weakly guarded Existential Datalog is the class "Warded Existential Datalog" [77]. This has been implemented and used in the Vadalog knowledge graph management system [28]. Warded TGDs offer a good trade-off between expressive power and computational complexity, with CQ answering proven to be in PTIME in data complexity [77].

Definition 16 (Dangerous Variables). Given a set of TGDs Σ , a variable X appearing in $body(\sigma)$, with $\sigma \in \Sigma$, is called harmless (wrt Σ) if at least one occurrence of X appears in a positive atom in $body(\sigma)$ and in $non_affected(\Sigma)$, which is the set of all non-affected positions (see Definition 14) in Σ . X is called harmful if it occurs only in affected positions of $body(\sigma)$. Finally, X is called dangerous if it is harmful and appears in $head(\sigma)$.

Building on the concept of dangerous variables, the Warded Existential Datalog can be formally introduced.

Definition 17 (Warded Existential Datalog). A TGD $\sigma \in \Sigma$ is *warded* if all the dangerous variables in $body(\sigma)$ appear in a single body atom, the *ward*, which shares only non-harmful variables with other body atoms. A set Σ is warded if all the TGDs in Σ are warded.

In the following, we provide an example of Warded Existential Datalog.

Example 6 (Warded Existential Datalog Program).

$$\begin{aligned} edb(X) &\rightarrow idb(X, Y). \\ idb(X, Y) &\rightarrow goal_1(X, Y). \\ idb(X, Y) &\rightarrow goal_2(X). \end{aligned}$$

A set of warded TGDs Σ has the property that after normalization steps (i.e., harmful joins elimination [21]), query answering on D under Σ can be performed over a variant of the oblivious chase. In such a variant, an instance is repaired by activating a TGD step only if the TGD produces a fact that is not isomorphic to one already in or has been already added to D . The execution of this variant of the chase for CQ answering under warded TGDs always leads to a finite sequence of chase steps. Figure 2.1 represents the syntactic containment of several languages belonging to the Datalog[±] family.

A notable restriction on warded TGDs is the fragment piece-wise linear and warded existential Datalog PWLW which combines the restrictions of the warded fragment with those imposed by piece-wise linear Datalog described in Definition 3. Query answering in the context of PWLW TGDs has been proved to be NLOGSPACE-complete in data complexity [31], whilst removing the wardedness condition leads to the undecidability of ontological query answering. Such data complexity results are the best lower bound possible for a reasoning language that can express powerful graph navigation—in fact, the graph reachability problem is in NLOGSPACE—and, at the same time, to model non-trivial ontological reasoning tasks.

Usefulness and advantages of Existential Datalog

Existential Datalog is useful in various contexts, some of which are here briefly mentioned and/or illustrated.

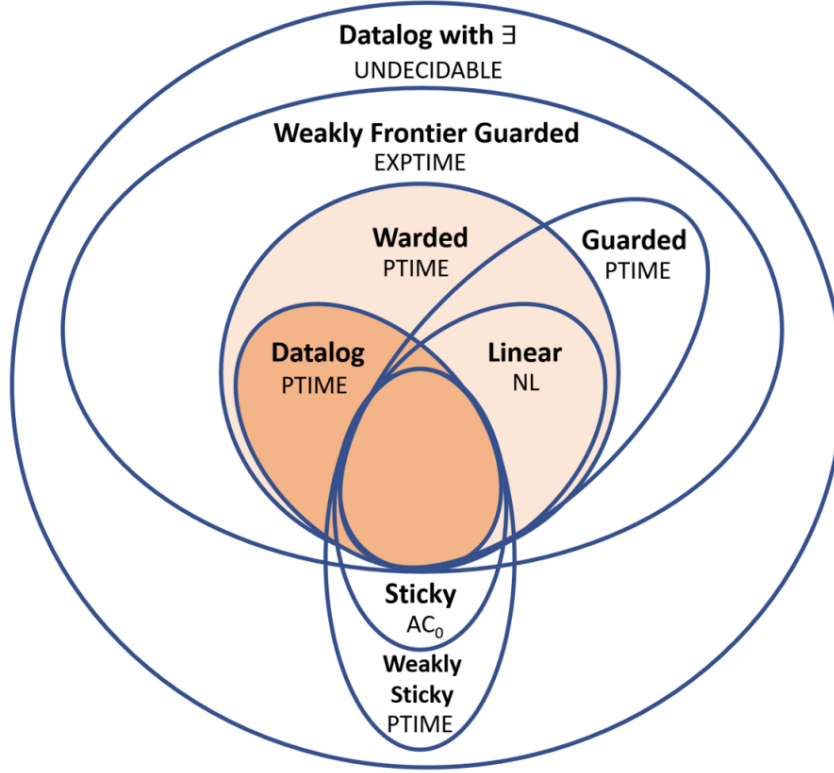


Figure 2.1: Syntactic containment of Datalog[±] languages. Annotations (non-bold) denote data complexity. All names not explicitly mentioning Datalog refer to the respective Datalog[±] languages (image taken from [30]).

Data Exchange

In data exchange, data from one database D_1 with a given schema has to be transferred to another database D_2 having a possibly different schema. This happens, for example, after a company acquisition, the data held in the database D_1 of the acquired company needs to be transferred into the database of the acquiring company. In such a case, D_1 may, for example, have a relation `employee` with attribute schema $(\text{LastName}, \text{FirstName}, \text{Address})$, and D_2 may have a relation `person` with attribute schema $(\text{Firstname}, \text{Lastname}, \text{Birthdate})$. The following Existential Datalog rule may be used for transferring the data from the `employee` relation to the `person` relation:

`employee(Lastname, Firstname, Address) → person(Firstname, Lastname, Null).`

By the above rule, those tuples in person that come from employee will have a (distinct) null value in the *Birthdate* field. The above rule is a so-called source-to-target TGD in the terminology of data exchange. In addition, possibly recursive existential rules (aka "TGDs") can also be used on the target database itself for further processing. In general, existential rules are an essential tool for data exchange.

Ontological Reasoning

It was already explained, how the ontological axiom $\text{house} \sqcap \neg\text{isolated} \sqsubseteq \exists\text{neighbor_property}$, with its above-stated meaning, can be translated into an Existential Datalog rule. More generally, it has been shown that description logics of the DL-Lite family, as well as extensions of the $\mathcal{EL}$ description logic such as $\mathcal{ELHI}$ can be translated straightforwardly into guarded Existential Datalog, which is a more expressive formalism. This shows that Existential Datalog is well-suited for ontological reasoning. In a similar way, Existential Datalog has been shown to be useful for reasoning about UML class diagrams and other types of class diagrams.

Duplicate Management.

It was also shown that Existential Datalog can be used for duplicate management in relational systems. Classical Datalog adheres to the set semantics, where a tuple can occur only once in a relation. However, considering duplicates is important for many practical applications, and therefore, commercial database management systems use by default bag semantics instead of set semantics. They usually keep duplicates by default and eliminate duplicates from query results only, when explicitly required. It has been shown that Existential Datalog can simulate bag semantics via set semantics. This can be done by adding a unique identifier (for example as the last argument) to tuples. Assume, for example, the binary relations r and s with schemas, respectively, (X, Y, Z) and (V, Y, W) and that the last column of both already consists of unique identifiers. Thus r may, for example, contain two tuples $r(a, b, 1)$ and $r(a, b, 2)$ that actually express a double occurrence of the tuple $r(a, b)$ in an equivalent database with a binary relation r that allows duplicates. Moreover, imagine that there are tuples $s(c, b, 3)$ and $s(c, b, 4)$ in the relation s , which reflects a pair of duplicates $s(c, b)$ in the corresponding "classical" binary relation. Then, in the classical setting, the natural join $t = r \bowtie s$ between the binary relations r and s (where duplicates

are allowed) would contain four instances of the tuple $t(a, b, c)$. This join can be simulated by the following Existential Datalog rule:

$$r(X, Y, Z), s(V, Y, W) \rightarrow t(X, Y, V, \text{Null}).$$

This rule creates new unique identifiers for the resulting t -tuples, for example, 5,6,7,8, and thus produces the tuples $t(a, b, c, 5)$, $t(a, b, c, 6)$, $t(a, b, c, 7)$, and $t(a, b, c, 8)$, that correctly correspond to the four occurrences of $t(a, b, c)$ in the join performed in a classical database system with duplicates.

Name invention

Another reason for existential variables in rule heads is name invention. In automated web data extraction, combining multiple objects into a single object is often necessary. Existential quantification can be used to name the new compound object. For example, on a web page, a user may see a table (displaying, say, articles and prices), but the HTML code underlying the web page actually implements this visual table by two HTML tables T_1 and T_2 that are of the same colour and contiguous so that the user sees only a single table T . An advanced data extraction program that analyzes the web pages may recognize the situation and create a new object for T , even though such an object is not explicitly present in the HTML code. In a simplified exemplary setting, the creation of T from T_1 and T_2 can be expressed by the following existential Datalog rules:

$$\begin{aligned} \text{html_table}(T_1), \text{html_table}(T_2), \text{colour}(T_1, C), \\ \text{colour}(T_2, C), \text{horiz_contiguous}(T_1, T_2) &\rightarrow \text{tablebox}(T, T_1, T_2). \\ \text{tablebox}(T, T_1, T_2) &\rightarrow \text{table}(T). \\ \text{tablebox}(T, T_1, T_2) &\rightarrow \text{contains_table}(T, T_1). \\ \text{tablebox}(T, T_1, T_2) &\rightarrow \text{contains_table}(T, T_2). \end{aligned}$$

The rules of this (simplified) example operate over a database that represents an extended parse tree (a.k.a. dom tree) of HTML documents. `html_table` is a predicate (whose definition is not listed here) such that `html_table( $t$ )` is true for all nodes of the parse tree that are the head nodes of an HTML table. `colour( $t, c$ )` is true if the table corresponding to t is coloured with colour c . `horiz_contiguous( $t_1, t_2$ )` is true if t_1 and t_2 are tables of the same height such that t_2 is directly to the right of t_1 and there is no line separating t_2 from t_1 . We may assume here that the

predicates `colour` and `horiz_contiguous( $t_1, t_2$ )` were computed from the parse tree and added to the EDB. The first rule then introduces a new value (via the existential quantifier) which stands for an object that is a tablebox and contains tables T_1 and T_2 .

Extensions of Existential Datalog

Most of the existing extensions to plain Datalog also work for Existential Datalog. A few are briefly mentioned here.

Stratification

The dependency graph $DG(\Sigma)$ for an existential Datalog program Σ is defined in exactly the same way as for a plain Datalog program, and also the definition of stratification is the same. The execution of the Chase for stratified Existential Datalog programs Σ on a database D can be concisely described as follows.

In case Σ belongs to a class of programs for which it is known that the Chase terminates, (e.g. weakly acyclic programs) one may execute the Chase stratum by stratum from the lowest to the highest. When stratum $i + 1$ is executed, all (finitely many) atoms previously computed for lower strata ($\leq i$) jointly act as database. This way, for each stratum, negative atoms in rule bodies can be evaluated just as for plain Datalog with stratified negation.

In case Σ belongs to a class of programs for which queries can be answered within a finite depth-bound of the chase (e.g. guarded or weakly frontier-guarded Existential Datalog), again, the Chase is executed stratum by stratum from the lowest to the highest. However, given that each stratum may now have an infinite result, the computation for each stratum is halted at the relevant depth bound (e.g., for weakly frontier-guarded programs, at derivation depth $f_{wg}(D, \Sigma)$, and thus also at depth $f_{wg}(D, \Sigma)$ of $WGCF(D, \Sigma)$). Again, each stratum gives rise to a finite number of atoms, and again, when stratum $i + 1$ is executed, all (finitely many) atoms previously computed for lower strata ($\leq i$) jointly act as database. This way, again, for each stratum, negative atoms in rule bodies can be evaluated just as for plain Datalog with stratified negation.

Finally, if Σ does not belong to a known syntactic class guaranteeing chase termination, nor belongs (or is known to belong) to any class with a guaranteed depth bound, then before computing the result for a stratum i , the computation of all lower strata ($< i$) must have terminated before "chasing" the rules of stratum i .

Of course, it is not guaranteed that this will happen, as there is no guarantee for Chase termination anyway.

Arithmetic Functions

Just as plain Datalog, Existential Datalog can be enriched with arithmetic functions, including also aggregate functions and comparison operators (that can be considered to be functions from numeric domains to Boolean). In general, this may destroy the finiteness of the Chase result, and is thus a potential danger, for example in case of a recursive rule $p(X) \rightarrow p(X + 1)$ applied to a database $\{p(0)\}$. However, in many cases, a careful or controlled use of arithmetic functions is useful and harmless.

Additional Declarations

In addition to the proper rules, various versions of Existential Datalog offer the possibility of making additional declarations of various types. The semantics of the permitted declarations has to be defined with each version of Existential Datalog. Some declarations may be optional while others mandatory. Examples of such declarations are: *input*(*r*) which specifies that a certain relation *r* is an input relation; *maxatoms*(*i*) or *maxlevel*(*k*) which specifies that the execution of the level-saturating Chase should be halted after *i* new atoms have been generated or after the *k* – *th* level has been computed, respectively. In some cases, instead of a constant *k*, such commands may also contain functional expressions that express a value, say, a function of the size of the input database. Another example which is useful, e.g., in the context of weakly frontier guarded Existential Datalog is a declaration such as *maxqsz*(*k*) which specifies that *k* is the maximum number of atoms any conjunctive query to be issued against the result *Chase*(*D*, Σ) of a program Σ and input database *D* (or, alternatively, the maximum size in terms of atoms of conjunctive queries for which the completeness and correctness of query-answering must be guaranteed). Other declarations that are allowed with certain versions of Existential Datalog are, for example, schema and/or type declarations. For example *Employee*(*FirstName* : *string*, *LastName* : *string*, *Salary* : *f32*), where *f32* indicates that the type of the third argument of the *Employee* relation must be a 32 bit floating-point number. Alternatively, if attribute names do not play a role, in some versions of Existential Datalog one may only specify the types of the arguments, for example by just stating *Employee*(*string*, *string*, *f32*) instead of the above. Every version of Existential Datalog may have its own set of possible additional declarations whose syntax and semantics should be clearly specified.

Skolem terms

As already explained, when "firing" existential rules, the Chase algorithm generates fresh new null values aka Skolem terms in those places of the newly generated atom that correspond to existentially quantified variables in the rule head. In some approaches, instead of meaningless null values (Skolem constants), structured Skolem terms are used that carry more provenance information. For example, let σ_i be the i -th rule of an Existential Datalog program Σ , and let $BV(\sigma) = \{X_1, \dots, X_k\}$ be the set of body variables of σ_i of which $\{Y_1, \dots, Y_h\} \subseteq BV(\sigma)$ are the non-existential head variables. Moreover, assume that the existentially quantified head variables are, in order, $Z_1, \dots, Z_e$. When σ_i is applied with a substitution θ , then, for Z_j , instead of just a meaningless new null value, a Skolem term $f[i, j](\theta(X_1), \dots, \theta(X_k))$ may be created and inserted at the position of Z_j in the atom created by the rule application. The advantage here is that such terms carry "provenance history": The term reveals from which data and by which rule and head variable it was derived. In naive implementations of the level-saturating Chase algorithms, the size of such Skolem terms may grow exponentially in the level. However, by keeping a table in which each term is identified by a separate name (that can be used as a subterm of further terms instead of the term proper) a system can avoid this drawback.

There are various implemented systems of Existential Datalog. A few examples are Vatalog [28], Graal [18], Shy DLV [105], and Vlog [41].

2.5 Knowledge Graphs (KGs)

Many of the notions mentioned in the current section follow those in [175, 58]. The term knowledge graph was originally used only to refer to Google’s Knowledge Graph [2], but it has been extended to other contexts over time. This led to several definitions being associated with the same term. As already reported in Chapter 1, we will refer to one of the most accepted and used definitions:

Definition 18. *A knowledge graph acquires and integrates information into an ontology and applies a reasoner to derive new knowledge.* [58]

Definition 18 is quite popular since it specifies the minimum and most common requirements for a KG to be considered as such.

There are three main categories of KGs: (standard) knowledge graphs (KGs); temporal knowledge graphs (TKGs), which associate a timestamp to tuples in the KG; and multi-modal knowledge graphs (MMKGs), which store several types of data such as images, videos and text. In the present discussion, we always refer to (standard) KGs.

From Definition 18 it is apparent that two tools are fundamental for KGs: a database to store the (large) amount of data of the KG and reasoning tools to derive the additional knowledge. A unified tool providing both these is referred to as Knowledge Graph Management System (KGMS), i.e., a Knowledge Base Management System (KBMS) capable of performing reasoning tasks over very large amounts of data. Moreover, a KGMS should provide methods and tools for data analytics and machine learning [26]. While there are graph databases, i.e. databases specifically designed to store and query graph data such as Neo4j [161], Amazon Neptune [150], JanusGraph [66], ArangoDB [87] or OrientDB [108], knowledge graphs are not strictly required to be stored on these. Alternatives include using relational databases such as MySQL [51], PostgreSQL [80], Microsoft SQL Server [50], Oracle Database [52] or SQLite [49], or document-oriented databases such as MongoDB [89], Couchbase [88], Amazon DocumentDB [149], Apache CouchDB [65] or RavenDB [107].

Storing knowledge graphs in a graph database offers several advantages:

- Knowledge graphs inherently have a graph structure, with entities represented as nodes and relationships as edges. Graph databases are designed to store and query graph structures, making them a natural fit for knowledge graphs.

2. Background and Related Work

- Knowledge graphs often have evolving and dynamic schemas. Graph databases are schema-less or schema-flexible, allowing for the addition of new node types and relationship types without significant database modifications.
- Graph databases excel at traversing relationships between entities. Queries that involve navigating through relationships in a knowledge graph can be more efficient in a graph database compared to traditional relational databases.
- Graph databases often use query languages like Cypher (for Neo4j) or SPARQL (for RDF databases), which are specifically designed for expressing graph patterns and traversing relationships. These languages are more intuitive for querying graph-structured data compared to SQL, which is optimized for tabular data. Also Datalog and its extensions are well-suited languages for expressing graph queries.
- Certain types of queries, such as finding paths between entities or identifying patterns in a graph, are inherently more efficient in a graph database. Graph databases can outperform relational databases in scenarios where the relationships between entities are a primary focus. The more a query requires traversing several relations, the more the performance gap is evident.

On the other hand, storing a knowledge graph in a relational database can offer other advantages:

- **Structured data:** Relational databases provide a structured way to organize and store data. This structure helps maintain the consistency and integrity of the knowledge graph.
- **Data integrity:** Relational databases enforce data integrity constraints, such as primary key and foreign key relationships, ensuring that data is accurate and consistent. This is particularly important when dealing with complex relationships in a knowledge graph.
- **Scalability:** Relational databases are designed to handle large amounts of data and can scale horizontally by adding more servers. This enables the storage and management of increasingly large knowledge graphs.

- **Security:** Relational databases provide robust security features, including access controls, authentication, and encryption. These features help protect the knowledge graph from unauthorized access and ensure data privacy.
- **ACID transactions:** Relational databases provide ACID (Atomicity, Consistency, Isolation, Durability) properties, ensuring that transactions are executed reliably and consistently. This is crucial for maintaining the integrity of the knowledge graph.
- **SQL Query Language:** Relational databases use SQL (Structured Query Language), which is a widely adopted and standardized language for querying and manipulating data. SQL offers powerful capabilities for complex joins, aggregations, and filtering operations. This enables efficient retrieval of information from the knowledge graph for certain operations.
- **Mature Technology:** Relational databases have been around for decades and are well-established technologies with robust features and mature tooling. They offer a wide range of functionalities, including transaction support, indexing, and query optimization. Moreover, there is a wider community of experts. This makes it easier to integrate and work with existing systems and applications.

In addition to the database type, the data format is not standardised. A potential choice could be the Resource Description Format (RDF) format, i.e. storing data in triples of the form:

(head, relation, tail)

where head and tail represent, respectively, the entities involved and relation the relationship between them. An example of this could be the triple *(Abe, parent, Bob)*, representing that *Abe* is the *parent* of *Bob*. Using the RDF data format facilitates the integration with graph databases such as Neo4j. On the other hand, constraining KGs to contain only data in the RDF format limits the integration of standard relational data which makes up a significant amount of existing data. On the contrary, the relational formalism is more flexible and would allow to store also RDF triples, although it does not allow for optimised query answering of graph structures. The choice of the specific data format and, as a consequence, type of database is taken considering the above-listed advantages and the specific requirements of the knowledge representation project. In the following, we provide a list of well-known knowledge graphs, most of which are freely available (out of this list, only the Google Knowledge Graph is proprietary) and only a few are

no longer maintained: Google Knowledge Graph [2], Wikidata [166], YAGO [158], Microsoft Academic Graph [153], Freebase [153], DBpedia [104], Knowledge Vault [56], OpenCyc [1], BabelNet [126], Probase [170], ConceptNet [154], WordNet [62], FrameNet [20]. These knowledge graphs are used in a variety of tasks, providing information to search engines, chatbots, virtual assistants, and other AI-powered systems. They contain vast amounts of structured and unstructured data about entities, concepts, and relationships between them. The symbolic representation of a KG is $KG = (\mathcal{E}, \mathcal{R}, \mathcal{F})$, where $\mathcal{E}$, $\mathcal{R}$, and $\mathcal{F}$ denote, respectively, the sets of entities, relations, and facts. In the RDF data format, $\mathcal{R}$, $\mathcal{E}$ and $\mathcal{F}$ correspond to sets of values appearing, respectively, in the *relation* entries, in the *head* and *tail* entries and in the set of triples (*head, relation, tail*) over all the dataset. In case a relational data format is used, $\mathcal{R}$, $\mathcal{E}$ and $\mathcal{F}$ correspond to sets of values appearing, respectively, in the tables' name, columns' name and tables' row.

Knowledge Graph Reasoning

Usually, KGs are represented under the open world assumption (OWA), which does not assume the knowledge in the KG to be complete. The opposite assumption is the closed world assumption (CWA), which expects missing knowledge to be false. Under the OWA, KGs are always considered incomplete. As a consequence, increasing their coverage is a constant effort over time. Another challenge in managing KGs is detecting and repairing incorrect knowledge. Knowledge Graph Reasoning (KGR) refers to applying reasoning methodologies to KGs to enhance the coverage and quality of the KG itself. The area of KGR can be divided into two major classes: KG smell detection (also known as KG denoising) and KG completion. The former refers to the set of techniques used to find and correct wrong knowledge, the latter refers to the set of techniques apt to enhance the comprehensiveness of the KGs. Given $KG = (\mathcal{E}, \mathcal{R}, \mathcal{F})$, the objective of a KG smell detection task is to detect facts of the form $f = (e_h, r, e_t)$, where $f \in \mathcal{F}$, e_h and e_t are in $\mathcal{E}$ and $r \in \mathcal{R}$, such that f is a faulty data, where we refer to a faulty data to a data point that is either partially or completely wrong. Given $KG = (\mathcal{E}, \mathcal{R}, \mathcal{F})$, the task of KG completion is to generate a new set of facts to be added in $\mathcal{F}$.

2.6 Inductive Logic Programming

Inductive logic programming (ILP) [124] is a subfield of machine learning (ML) derived from first-order logic that uses logic programming as a uniform representation. ILP aims to learn logical rules (such as Datalog rules) from data, using various techniques such as inverse resolution, generalization, abduction and entailment. The most relevant ILP-based KGR methods are First-Order Inductive Learning (FOIL), Relation Path Sampling and Association Rules Mining. In the following, we will describe Association Rules Mining techniques and briefly introduce related techniques such as First-Order Inductive Learning and Relation Path Sampling.

Association Rules Mining

The notions introduced in this section refer to those in [84]. In the first half of the 1990s, an approach called association rules mining (ARM) was proposed. This data mining technique is used to uncover relationships between different items in a dataset (i.e., data patterns), such as items that are frequently purchased together. It is a type of unsupervised learning, so it does not require labelled data. This type of analysis has been historically used to identify customer buying patterns for the purpose of informing marketing strategies and product placement.

Example 7. Assume a database D with unnormalized schema (for the definitions of unnormalized and normalized schemas, refer to [47])

$$\text{transaction}(\text{user}, \text{timestamp}, [\text{item}_1, \dots, \text{item}_n])$$

containing data about transactions of customers. To design marketing strategies, it is crucial to understand which products (items) are bought together to profile customers and recommend related products. Using ML techniques (such as association rules mining), it is possible to discover patterns such as

$$\text{item}_x, \text{item}_y \rightarrow \text{item}_z$$

, where $x, y, z \leq n$. Such a rule indicates that when users buy item_x and item_y , they often buy also item_z . To better clarify the relevance and reliability of the rule, metrics such as support and confidence are additionally provided with the rules.

Over the years, ARM techniques have also been widely used in other areas such as web usage mining [100], bioinformatics [54, 125], retail [144], finance [86], healthcare [48, 156] and education [142, 70].

The first and most-used algorithm for mining association rules is the Apriori algorithm proposed by Agrawal and Srikant in 1994 [8]. This bottom-up approach uses a set of rules to identify frequent itemsets (patterns). The name of the algorithm is based on the fact that the algorithm uses "prior knowledge" of frequent itemset properties. The algorithm works by first identifying all items that appear in the dataset and then finding all combinations of items that appear together, i.e., the itemsets. Then, it calculates the support of each itemset and discards those whose value is below a certain user-defined threshold. The support is a representation of the usefulness of an itemset of attributes and is computed as:

$$S(\{A, B\}) = P(A \cap B) = \frac{\text{number of data points containing both } A \text{ and } B}{\text{total number of data points}} \quad (2.1)$$

, where A and B are separate item sets that occur in the same data point. A support value of, say, 0.02 would indicate that, of all the records mined, the attributes in the itemset under analysis appear together in at least 2% of the records. Once all the itemsets of maximum size are found, the association rules are created from these. For each itemset of the form $\{a(v_1), \dots, a(v_n)\}$, n rules are created using one of the item $a(v_i)$ as the head of the rule and the remaining items $\{a(v_1), \dots, a(v_n)\} \setminus \{a(v_i)\}$ as the body of the rule, where $1 \leq i \leq n$. Then, for each rule, a confidence score is used to measure the strength of the rule, computed as:

$$C(A \rightarrow B) = P(B|A) = \frac{S(\{A, B\})}{S(A)} = \frac{\text{number of data points containing } A \text{ and } B}{\text{number of data points containing } A} \quad (2.2)$$

, where A and B are separate item sets that occur in the same data point. Confidence is a representation of the certainty of an association rule. A confidence of, say, 0.6 for a given association rule would indicate that 60% of the time, the body of the rule is found true in the records of the dataset, and the head is true. Rules that satisfy both a minimum support and confidence threshold are called strong rules.

The Apriori algorithm is just one of many algorithms used for association rule mining [97, 73]. Other algorithms include the Eclat algorithm [173], which is a depth-first approach, and the FP-Growth algorithm [85], which is a more efficient version of the Apriori algorithm. Each algorithm has its own strengths and weaknesses, and the choice of algorithm depends on the dataset and the desired results. In addition to the algorithms used for association rule mining, several metrics are used to measure the strength of the association between items. Besides support and confidence, described in the previous paragraph, the lift

measure can be used. This is defined as:

$$lift(A \rightarrow B) = \frac{S(\{A, B\})}{S(A) * S(B)} \quad (2.3)$$

The lift value is the ratio of the confidence of the itemset to the expected confidence if the items were independent.

Limitations

Current systems making use of association rules mining present several limitations. Here, we briefly summarise them:

- State-of-the-art algorithms do not allow to mine association rules from fuzzy sources, where tuples or attributes of the tuples are associated with a truth value $0 \leq T \leq 1$. This forces the user to threshold on these, i.e. transforming
- Employed algorithms are often computationally intensive and require large amounts of data to be processed. This can be a problem for smaller datasets, as the ARM algorithms may produce association rules of low quality due to the lack of data. Additionally, the algorithms may not be able to detect subtle patterns in the data if those patterns do not appear often in the dataset, minimising the number of association rules produced.
- Conventional approaches using association rule mining techniques lack scalability features. Association rule mining algorithms are not very efficient when dealing with large datasets because the number of possible rules that can be generated increases exponentially with the size of the dataset.
- Current approaches might produce association rules with low accuracy. Association rule mining algorithms alone are not very accurate in predicting the strength of the association between items. This is because the algorithms focus on predicting all the possible association rules based on some heuristics.
- Systems based on association rule mining often lack interpretability. This is partially due to Association rule mining algorithms not being very interpretable. On top of this, the huge amount of rules usually produced by such systems is hard to review. This is partially due to the nature of the association rules. However, the impact of this could be reduced by using data visualization techniques.

- Usually, the integration of the human component with the business processes is an overlooked aspect.
- There is no system providing a unified approach to address all of the criticalities mentioned. Applying the existing and newly proposed techniques requires an orchestration component and employing self-adjusting algorithms not found in the current literature.

As such, it is important to be aware of these limitations when considering using association rule mining algorithms to uncover hidden patterns in datasets.

Further ILP Techniques

First-Order Inductive Learning

(FOIL) is a rule-based learning algorithm used to infer new rules from existing textual data. It uses a four-step process to learn rules: (1) identification of relevant classes and their corresponding instances, (2) discovery of interclass relations, (3) acquisition of inference rules based on these relations and determination of their confidence scores, and (4) performing logical reasoning and search for facts, with the confidence of each fact estimated using a Markov network [11]. Modifications of the FOIL technique have been proposed to address specific issues. N-FOIL [98] introduces mechanisms to reduce the number of instances in each iteration and increase the diversity of the learned rules. T-FOIL [98] reduces the number of model parameters. K-FOIL [99] integrates kernel learning techniques, improving both accuracy and efficiency. QuickFOIL [174] adopts a top-down greedy ILP search and integrates a scoring function and pruning strategy to guide the search process. As a consequence, it is capable of discovering higher-quality rules.

FOIL-based approaches have been proven effective in performing efficient reasoning within small-scale KGs. Unfortunately, the computational complexity of these algorithms is exponential in the number of entities and relations. Applying these approaches to medium or big-sized KGs would result in a slower computation of more inaccurate rules. Moreover, FOIL methods commonly rely on the availability of negative examples which are hard to obtain under the open-world assumption (OWA) of the KGs.

Relation Path Sampling

is a rule-mining technique which aims to reduce the dimension of the search space. In order to do so, it uses as input for its learning phase a subset of triples from the

KG which are sampled to have a direct link to the target relation. In the following, we will briefly introduce some approaches that fall into this category. RuleN [118] leverages a dual random sampling to identify two types of rules called P_n -type and C -type. These are of the form:

$$\begin{aligned} P_n &: s_1(x_1, x_2), \dots, s_n(x_n, x_{n+1}) \rightarrow r(x_1, x_{n+1}) \\ C &: r(x, y) \rightarrow r(x, a) \end{aligned}$$

C-NN [63] makes use of entity clustering to reduce the computational costs at training time. To do so, it makes use of a K-nearest neighbors (KNN) algorithm on the space of the entities. AnyBURL [117] is a bottom-up iterative process that finds longer rules in each iteration which ensures computational efficiency. SAFRAN [129] uses denoising techniques to detect and cluster redundant rules, which ensure the high quality of the mined rules.

2.7 Fuzzy Logics

One way to interpret uncertain observations is expressing levels of confidence in a fact in terms of degrees of truth as in fuzzy logics [83]. That is, a fact that is considered to be true to a certain degree, in accordance with the level of confidence in the observation. Fuzzy logic is a form of many-valued logic based on the idea of fuzzy sets, introduced by Zadeh in the 1960s [172]. Fuzzy logic is a branch of logic used to handle data that are inherently not crisp, i.e., information that is not entirely true or false. An example of such kind of information is weather forecast data. The truth values of data points in fuzzy logic are real numbers in the interval $[0, 1]$. Fuzzy logic can be used to model complex systems that are difficult to model using traditional methods such as in boolean logic where truth values are either 0 or 1. Fuzzy sets are imprecise statements such as tall or short, which are relative but not absolute measures, to which data points can be assigned to by using membership functions [91]. The most common membership functions (MFs) are:

- Triangular MF, shown in Figure 2.2(a), and defined as:

$$\mu_A(x) = \begin{cases} 0, & \text{if } x \leq a \\ (x - a)/(b - a), & \text{if } a < x \leq b \\ (x - c)/(b - c), & \text{if } b < x < c \\ 0, & \text{if } x \geq c \end{cases}$$

where the range "a to c" depicts the support of the fuzzy set and b is the unique point in the range with the highest membership function value. The attributes of this simple MF are represented by its single element with the maximum value of membership grade and the linear variation of the membership grade moving away from this.

- Trapezoidal MF, shown in Figure 2.2(b), and defined as:

$$\mu_A(x) = \begin{cases} 0, & x < a \\ (x - a)/(b - a), & a \leq x \leq b \\ 1, & b < x < c \\ (d - x)/(d - c), & c \leq x \leq d \\ 0, & x \geq d \end{cases}$$

where the support of the fuzzy set is framed between a and d . Like the triangular MF, the trapezoidal MF also has a linear variation of the membership grade, but it is characterised by the range " b to c " having the maximum value of membership grade.

- Gaussian MF, shown in Figure 2.2(c), and defined as:

$$\mu_A(x) = e^{-(x-c)^2/\sqrt{2}\sigma}$$

where, σ denotes the standard deviation, and c is hyperparameter governing the shape of the MF. The Gaussian MF is characterized by a non-linear variation of the membership grade, in addition to not having hard boundaries.

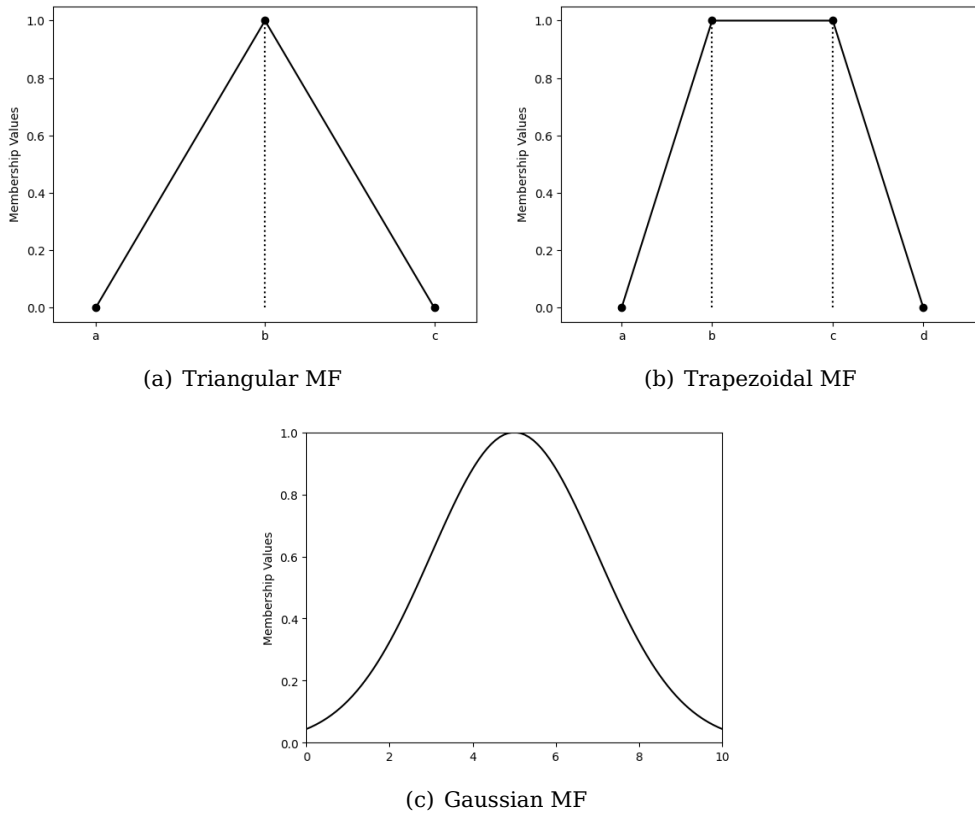


Figure 2.2: Common Membership Functions

Each data point is assigned to one or more fuzzy sets and associated with a degree of belonging to the membership function (also referred to as truth degree) in the interval $[0, 1]$.

Algebraic t-Norms

In order to combine conjunctions of fuzzy statements, it is possible to use t-norms. These have been applied to fuzzy logic by replacing the standard set of operators with algebraic t-norms, giving birth to fuzzy logic fragments which have usually been named after the t-norm operator they build upon. A notable example is the Gödel fuzzy logic based on the set of operators proposed by Zadeh [135]:

$$AND(x, y) = MIN(x, y) \quad (2.4)$$

$$OR(x, y) = MAX(x, y) \quad (2.5)$$

$$NOT(x) = 1 - x \quad (2.6)$$

Other commonly adopted fuzzy logics based on t-norms are the Łukasiewicz fuzzy logics [83], which employs the following set of operators:

$$AND(x, y) = MAX(0, x + y - 1) \quad (2.7)$$

$$OR(x, y) = MIN(1, x + y) \quad (2.8)$$

$$NOT(x) = 1 - x \quad (2.9)$$

and the product fuzzy logic [83], which employs the following set of operators:

$$AND(x, y) = x * y \quad (2.10)$$

$$OR(x, y) = MAX(x, y) \quad (2.11)$$

$$NOT(x) = 1 - x \quad (2.12)$$

Since its introduction, fuzzy logic has become increasingly popular as a tool for solving complex problems. It has been used in many fields such as AI, search engines, engineering, medicine, economics, and robotics. One of the advantages of using fuzzy logic to handle uncertainty, rather than using probabilistic systems (in the cases where probabilities are not required), relies on the simplicity of the formalism that can achieve real-time performances to make fast decisions, which is useful for applications such as robotics.

There have been a wide variety of approaches to combining logic programming with fuzzy logic. Pioneering work in this area by [5], [57], and [165] introduced a number of early foundational ideas for fuzzy logic programming. The first

also considers fuzziness in the object domain (expressing the possibility that two constants may actually refer to the same object), via fuzzy similarity relations on the object domain. This idea was further expanded in the context of deductive databases by [90].

One limiting factor of these approaches is the reliance on the Gödel t-norm (see, [135]) as the basis for many-valued semantics. The Gödel t-norm of two formulas ϕ, ψ is the minimum of the respective truth degrees of ϕ and ψ (see 2.4). While this simplifies the model theory of the underlying logics, the resulting semantics are not ideal for applications in which uncertain observations from AI systems are combined with large certain knowledge bases.

2.8 Neuro-Symbolic (NeSy) Reasoning

An alternative way to interpret uncertain observations is to consider those as the likelihood of the fact being true, i.e., the level of confidence in a fact is interpreted as a probability of the fact holding true. Such an interpretation has been widely studied in the context of Markov Logic Networks [139], Problog [137] and probabilistic databases [159, 42]. However, generally, such formalisms make strong assumptions on the pairwise probabilistic independence of all tuples which can be difficult to satisfy. An approach to probabilistic reasoning in Datalog[±] based on the chase procedure was proposed very recently by [29].

A different approach to tackle uncertain observations is Probabilistic Soft Logic (PSL), introduced by [17] as a framework for fuzzy reasoning with semantics based on Łukasiewicz logic. PSL allows for negation in rule bodies as well as disjunction in the head, in addition to certain forms of aggregation. Semantically, it treats all rules as soft constraints and aims to find interpretations that satisfy rules as much as possible. Similar ideas have also been proposed in fuzzy description logics, see e.g., [34, 36, 157], or [109] for an overview of fuzzy DLs.

In the following, we are going to give a more detailed overview of the relevant methods proposed in the literature.

Markov random fields

The definition in this section follows those in [17].

Markov random fields (MRFs), also called *Markov networks* (MNs), are models for the joint distribution of sets of variables introduced by Pearl in the 1989 [132]. These can be utilized to capture how a certain domain behaves. An MRF is composed of an undirected graph G and a set of potential functions ϕ . The graph has a node for each variable, and the model has a potential function for each clique in the graph. A potential function is a nonnegative real-valued function of the state of the corresponding clique.

Definition 19. Let $\mathbf{x} = (x_1, \dots, x_n) \in \chi$ be a vector of random variables and let $\phi = (\phi_1, \dots, \phi_m)$ be a vector of potentials where each potential $\phi_j(\mathbf{x})$ assigns configurations of the variables a real-valued score. Also, let $\mathbf{w} = (w_1, \dots, w_m)$ be a vector of real-valued weights. Then, a *Markov random field* is a probability distribution of the form

$$P(\mathbf{x}) \propto \exp(\mathbf{w}^T \phi(\mathbf{x}))$$

As already thoroughly explained in Section 2.3, an elegant formalism to express the constraints present in a domain is through the definition of first-order logic clauses (also interpretable as rules) $C = \{C_1, \dots, C_m\}$. It is then possible to translate the set C to a set of potentials $\Phi = \{\phi_1, \dots, \phi_m\}$ assigning $\phi_j(\mathbf{x})=1$ when $\mathbf{x}$ satisfies C_j , 0 otherwise. To calculate this assignment, it is possible to use the formula:

$$\phi_j(\mathbf{x}) = \min \left\{ \left(\sum_{i \in I_j^+} x_i + \sum_{i \in I_j^-} (1 - x_i) \right), 1 \right\} \quad (2.13)$$

where I^+ is the set of positive variables in C_j and I^- is the set of negative ones. Once the set Φ has been computed, it is possible to find a most probable assignment to the variables through maximum a posteriori (MAP) inference. This can be expressed with the linear program:

$$\arg \max_{\mathbf{x} \in \{0,1\}^n} P(\mathbf{x}) = \arg \max_{\mathbf{x} \in \{0,1\}^n} \mathbf{w}^\top \boldsymbol{\phi}(\mathbf{x}) \quad (2.14)$$

where $\mathbf{w}$ is the set of weights that represent the relative importance of potentials and can be assumed as given or trained afterwards.

Although the linear program in Equation 2.14 is intractable, it allows for convex programming relaxations. The described formalism can be used to translate a plain Datalog program Σ over a schema S , where each rule σ_i can be translated into the logical clause C_i , which can be translated to the potential ϕ_i . Using this, it is possible to express a Datalog program with soft rules instead of hard ones, expressing the uncertainty over the possible worlds in which a certain fact is considered true or false. An approach that makes use of MRF to embed a first-order knowledge base is shown in [139].

Hinge-loss Markov random fields

A recent approach to embedding probabilities into logic models makes use of an undirected probabilistic graphical model called *hinge-loss Markov random field* (HL-MRF) [17]. HL-MRFs represent an extension of the *discrete Markov random field* (MRF) model. In this subsection, we will introduce how HL-MRF manages to capture both the structure and the uncertainty in different domains, such as, e.g. in the presence of neural networks or machine learning outputs. The definitions presented here follow those in [17].

Hinge-loss Markov random fields make use of *hinge-loss functions* in order to extend the domain from the discrete variables $\mathbf{x} = (x_1, \dots, x_n) \in \chi$ with joint domain $\{0,1\}^n$ used in MRFs to continuous variables $\mathbf{y} = (y_1, \dots, y_n)$ with joint

domain $[0, 1]^n$.

To embed logical clauses into the continuous domain, it is possible to interpret them through Łukasiewicz Logic [94] which translates the Boolean logic operators (as previously defined through Equation 2.7).

With such interpretation, it is possible to assign soft values to the literals in clauses and embed a clause C_j in potentials of the form:

$$\phi_j(\mathbf{y}) = \min \left\{ \left(\sum_{i \in I_j^+} y_i + \sum_{i \in I_j^-} (1 - y_i) \right), 1 \right\}$$

where I^+ is the set of positive variables in C_j and I^- is the set of negative ones. For convenience, the above equation can be rewritten to:

$$\phi_j(\mathbf{y}) = \max \left\{ \left(1 - \sum_{i \in I_j^+} y_i - \sum_{i \in I_j^-} (1 - y_i) \right), 0 \right\} = \max \{ l(\mathbf{y}), 0 \}$$

where $l(\mathbf{y})$ represents the distance to the satisfaction of the clause C_j through a hinge-loss function.

Leveraging the equation for the potential of a clause, it is possible to address the problem of MAP inference using the inference objective:

$$\arg \min_{\mathbf{y} \in [0,1]^n} \sum_{C_j \in \mathcal{C}} w_j \max \{ l(\mathbf{y}), 0 \}$$

or, to better reflect the relative importance between the clauses, it is possible to use a squared hinge-loss function, with the following inference objective:

$$\arg \min_{\mathbf{y} \in [0,1]^n} \sum_{C_j \in \mathcal{C}} w_j \left(\max \{ l(\mathbf{y}), 0 \} \right)^2$$

HL-MRFs can also be generalized to use arbitrary linear functions $l(\mathbf{y})$, but in the context of this document, we will not comment on this generalisation; therefore, by $l(\mathbf{y})$, we always refer to either hinge-loss or squared hinge-loss functions, as shown in the previous formulas.

The definition of an HL-MRF builds on the constrained hinge-loss energy function which is defined as:

Definition 20. Let $\mathbf{y} = (y_1, \dots, y_n)$ be a vector of n variables and $\mathbf{x} = (x_1, \dots, x_n)$ a vector of n' variables with joint domain $D = [0, 1]^{n+n'}$. Let $\phi = (\phi_1, \dots, \phi_m)$ be a vector of m continuous potentials of the form

$$\phi_j(\mathbf{y}, \mathbf{x}) = (\max \{ l_j(\mathbf{y}, \mathbf{x}), 0 \})^{p_j}$$

where $1 \leq j \leq m$, l_j is a linear function of y and x , $p_j \in \{1, 2\}$. Let $c = (c_1, \dots, c_r)$ be a vector of r linear constraint functions associated with index sets denoting equality constraints E and inequality constraints I , which define the feasible set

$$\tilde{D} = \left\{ (\mathbf{y}, \mathbf{x}) \in D \mid c_k(\mathbf{y}, \mathbf{x}) = 0 \forall k \in E \text{ and } c_k(\mathbf{y}, \mathbf{x}) \leq 0 \forall k \in I \right\}$$

For $(\mathbf{y}, \mathbf{x}) \in D$, given a vector of m non-negative free parameters, i.e., weights, $\mathbf{w} = (w_1, \dots, w_m)$, a *constrained hinge-loss energy function* f_w is defined as

$$f_w(\mathbf{y}, \mathbf{x}) = \sum_{j=1}^m w_j \phi_j(\mathbf{y}, \mathbf{x})$$

Now, it is possible to formally define an HL-MRF.

Definition 21. A *hinge-loss Markov random field* P over random variables $\mathbf{y}$ and conditioned on random variables $\mathbf{x}$ is a probability density defined as: if $(\mathbf{y}, \mathbf{x}) \notin \tilde{D}$, then $P(\mathbf{y}|\mathbf{x}) = 0$, otherwise

$$P(\mathbf{y}|\mathbf{x}) = \begin{cases} 0 & \text{if } (\mathbf{y}, \mathbf{x}) \notin \tilde{D} \\ \frac{1}{Z(\mathbf{w}, \mathbf{x})} \exp(-f_w(\mathbf{y}, \mathbf{x})) & \text{if } (\mathbf{y}, \mathbf{x}) \in \tilde{D} \end{cases}$$

where

$$Z(\mathbf{w}, \mathbf{x}) = \int_{\mathbf{y} | (\mathbf{y}, \mathbf{x}) \in \tilde{D}} \exp(-f_w(\mathbf{y}, \mathbf{x})) d\mathbf{y}$$

Probabilistic Soft Logic

Probabilistic Soft Logic (PSL) [17] is a probabilistic programming language based on the HL-MRF class of probabilistic graphical models, described in Section 2.8. The definitions in this section follow those proposed in [17].

Definition 22. A *PSL program* is a set of rules, each of which is a template for hinge-loss potentials or hard linear constraints. When grounded over a base of ground atoms, a PSL program induces an HL-MRF conditioned on any specified observations.

PSL allows for two kinds of rules: logical rules and arithmetic rules. The former are first-order logic rules, extended with an optional weight and an optional squared symbol of the form

$$\begin{aligned} w : R_1(\bar{u}_1) \ \&\& \dots \ \&\& R_n(\bar{u}_n) \rightarrow R_p(\bar{u}_p) \ \&\& \dots \ \&\& R_q(\bar{u}_q) \\ R_1(\bar{u}_1) \ \&\& \dots \ \&\& R_n(\bar{u}_n) \rightarrow R_p(\bar{u}_p) \ \&\& \dots \ \&\& R_q(\bar{u}_q) \end{aligned}$$

, where the first rule is a weighted rule with weight $w \geq 0$ and the second one is unweighted. The reported notation is the one used in PSL syntax, where $\&\&$ represents a conjunction.

An arithmetic rule is a rule that, instead of using logical operators, uses arithmetic operators. The arithmetic rules can also be weighted or unweighted as logical rules. Their syntax is not relevant to this document; therefore, it will not be presented.

PSL requires four sets of inputs to be provided. The first one is the set of *closed predicates* C , i.e., the completely observed atoms. The second one is the set of *open predicates* O , i.e. the atoms that may be unobserved. The third one is the *base* A , i.e. the set of all ground atoms under consideration. The last one is a function $\theta : A \rightarrow [0, 1] \cup \{\emptyset\}$ that maps ground atoms in the base to either an observed value in $[0, 1]$, or a symbol $\emptyset$ indicating an unobserved ground atom. Once the input is defined, each ground atom $a \in A$ is associated with a random variable in the domain $[0, 1]$. If $\theta(a) = \emptyset$, then $a \in \mathbf{y}$, otherwise $a \in \mathbf{x}$, where $\mathbf{y}$ is the set of free variables and $\mathbf{x}$ is the set of observations for $P(\mathbf{y}|\mathbf{x})$. Then, each rule is grounded using every combination of ground atoms in A . This produces a set of grounded rules G . Each $g_i \in G$ is then associated with a potential ϕ_i using Definition 20. Finally, it is possible to find a most probable assignment for the free variables $\mathbf{y}$, using an approximated algorithm for MAP inference.

PSL is capable of modelling arbitrary probabilistic relationships among Boolean variables, as Markov logic networks [139] do.

Although very interesting, PSL has some crucial limitations:

1. it does not provide any inference engine, forcing the user to provide the full set of atoms A to consider;
2. it performs the full grounding, which is unfeasible to apply for real systems;
3. it produces unintuitive and unnecessary outputs;
4. it provides a limited semantic.

The first point of limitations follows directly from the description of the inputs. PSL is not an inference engine, and it assumes that logic inference will be addressed with external tools.

The second statement comes from the above-described grounding procedure which grounds every rule with every combination of ground atoms in A . When applied to real systems handling databases containing data ranging over large

domains, this is unfeasible and leads to the subsequent statement on limitations, for which we provide a brief example to highlight the problems. Consider the following program Σ :

```

ancestor(X, Y) && ancestor(Y, Z) → ancestor(X, Z).
ancestor(Alice, Bob).
ancestor(Bob, Charlie).
ancestor(Charlie, Dora).
ancestor(Alice, Eleonor).

```

Solving for the query $\text{ancestor}(\text{Eleonor}, \text{Bob})?$, the full grounding performed by PSL produced, among others, such grounded rules:

```

ancestor(Eleonor, Bob) && ancestor(Bob, Charlie) → ancestor(Eleonor, Charlie).
ancestor(Eleonor, Charlie) && ancestor(Bob, Dora) → ancestor(Eleonor, Dora).

```

It is possible to notice how the transitive closure described by Σ would never produce $\text{ancestor}(\text{Eleonor}, \text{Charlie})$ or $\text{ancestor}(\text{Eleonor}, \text{Dora})$ using standard logical inference.

This example highlights a criticism of the ground procedure adopted in PSL. Moreover, solving a query answering task over a grounded atom requires specifying it in the input set. As the grounding procedure used by PSL uses all the inputs provided, it uses the head of the query as a possible body for other rules. Therefore, the direction implied in the rules is not considered and $\text{ancestor}(\text{Eleonor}, \text{Charlie})$ or $\text{ancestor}(\text{Eleonor}, \text{Dora})$ are considered plausible results. Such a grounding procedure introduces an unnecessary computational overhead and, most importantly, leads to less accurate results.

PSL also has some semantic limitations, such as:

- PSL cannot handle negation, as it requires introducing additional variables and constraints
- PSL cannot capture non-monotonic reasoning, such as default rules or exceptions, as it assumes that more evidence always increases the truth value of a formula.
- PSL cannot represent higher-order logic.

Markov Logic Network

A Markov Logic Network (MLN) [95, 139] is a probabilistic graphical model. This approach combines logic and probability by attaching weights to first-order formulas. To achieve this, MLNs leverage the uncertainty handling of MRFs defined in Section 2.8.

An MLN consists of a set of weighted formulas in first-order logic and a finite set of constants that represent the objects in the domain. The formulas shape how the domain behaves, while the weights reflect the confidence of each formula. Given a set of constants C and formulas Σ , an MLN defines a ground MRF containing one pair feature-weight for each possible grounding of $\sigma \in \Sigma$ using the constants in C .

Inference in MLNs consists of computing the probability of a query formula given some evidence. This can be done by using Markov chain Monte Carlo (MCMC) methods [127], such as Gibbs sampling or Metropolis, to sample from the posterior distribution. Alternatively, approximate inference methods, such as belief propagation [171] or variational inference [33], can be used to estimate the marginal probabilities of the query nodes.

Weight learning can be done by maximizing the data's likelihood or pseudo-likelihood in a knowledge base, using gradient-based methods or convex optimization techniques. Formula learning can be done by using inductive logic programming methods, such as FOIL introduced in Section 2.6, to generate and evaluate candidate clauses based on their weight and coverage. Alternatively, weight learning can also be performed by optimizing a loss function that measures the prediction error on the data.

Unfortunately, MLNs have several limitations:

- they can be computationally intensive to perform inference and learning on large KBs as they require grounding every formula with every constant in the domain;
- they are limited in tasks such as link prediction, as they rely on existing formulas;
- they demonstrate little interpretability, as they involve complex interactions between logic and probability;
- the inference in MLNs is usually difficult due to the complicated graph structures;

ProbLog

ProbLog [137] is a probabilistic extension of the language Prolog. It addresses the problem of managing probabilities in Prolog programs. This challenge has been tackled previously by other researchers [68, 123, 134, 147], although none of these approaches has been adopted in the implementation of real systems. Possible causes of this are the assumptions imposed to facilitate the computation or an inefficient inference engine.

A ProbLog program Σ specifies a probability distribution over all the possible non-probabilistic subprograms of Σ . The success probability of a given query is then defined as the probability that it succeeds in these subprograms.

Definition 23. A *ProbLog program* Σ is a finite set of rules $\sigma_1, \dots, \sigma_n$ and facts $e_1, \dots, e_m$, with $n \geq 1$ and $m \geq 1$. Each rule σ_i and fact e_j , with $i \in [0, n]$ and $j \in [0, m]$, is also labelled with a probability p .

A distribution over a subprogram Σ' of a ProbLog program Σ , with $\Sigma' \subseteq \Sigma$ is defined as:

$$P(\Sigma'|\Sigma) = \prod_{\sigma_i \in \Sigma'} p_i \prod_{\sigma_i \in \Sigma \setminus \Sigma'} (1 - p_i)$$

Definition 24. The *success probability* $P(q|\Sigma)$ of a query q over a program Σ is defined as:

$$\begin{aligned} P(q|\Sigma') &= \begin{cases} 1 & \exists \theta : \Sigma' \models q\theta \\ 0 & \text{otherwise} \end{cases} \\ P(q, \Sigma'|\Sigma) &= P(q|\Sigma') \cdot P(\Sigma'|\Sigma) \\ P(q, \Sigma) &= \sum_{\Sigma' \subseteq \Sigma} P(q, \Sigma'|\Sigma) \end{aligned}$$

From Definition 24 it is possible to notice that the success probability assumes the query and the subprograms as independent. Moreover, in order to compute the success probability p of a program Σ , it is necessary to generate all the subprograms Σ' . This would be unfeasible for real programs and, to overcome this problem, an approximation algorithm has been specially devised for ProbLog.

First, the algorithm transforms the query into a boolean monotone Disjunctive Normal Form (DNF) formula of binary random variables. Then, as the problem of computing the probability of boolean monotone DNF formula is NP-hard [163],

ProbLog uses the aforementioned approximated algorithm to compute an answer more efficiently. Details on the algorithm can be found in [137]

This approach has produced interesting results, showing some scalability features. Nevertheless, it is not capable of managing programs with more than 100000 formulae [137]. Moreover, it cannot guarantee termination, as nor Prolog guarantees it. A probabilistic version of Datalog could easily be envisioned. This would allow to ensure termination, other than taking advantage of the mature algorithms for scalability already developed in Datalog. Nevertheless, the problem of computing the success probability of a query would remain intractable as the need to generate all the possible subprograms would remain in place.

CHAPTER 3

Reasoning In Many-Valued Settings

Recent advances in the broad area of AI offer new methods for predicting and correcting data. However, auto-generated data is inherently uncertain, and its integration into knowledge bases without human validation presents risks and introduces new challenges. Other uncertain data sources include data gathered from crowd-sourcing applications or automated web crawlers.

Relying solely on human efforts to maintain large knowledge bases is unsustainable. Modern applications demand continually updated and on-demand knowledge, often at the expense of data accuracy. If improperly managed, data uncertainty can lead to flawed reasoning and decision-making, with severe implications in critical fields like healthcare, finance, and security.

Current approaches often overlook the uncertain factor or use thresholds, oversimplifying the data and losing potential insights from its nuances.

Acknowledging the current methodologies' limitations (Section 2.8), this chapter explores uncertainty management in logical reasoning. We suggest extensions to Datalog and Datalog[±] for many-valued settings and discuss their application. This chapter is based on two previous papers, with Section 3.1 addressing the shortcomings in [101] and Section 3.2 building on the methodologies and results in [32].

3.1 MV-Datalog and MV-Datalog[±]

Datalog has become a well-established reasoning formalism thanks to its declarative approach which allows a steep learning curve for experts. Moreover, its popularity amongst the research community brought solid theoretical foundations, fostering a broad adoption for real-world applications.

Nevertheless, there is no suitable formalism proposed in the literature when it comes to reasoning using uncertain data.

Imagine a database containing patients' medical records. Traditionally, in a binary logic framework, each data point in this database would be categorised as either true or false, providing a definitive answer to questions such as "Does the patient have cancer?" However, the reality of medical data is often far from binary, especially when considering automated diagnostic tools.

In modern healthcare systems, diagnostic processes increasingly rely on automated algorithms and machine learning models to analyse complex medical data. These tools not only provide diagnoses but also offer associated uncertainty metrics, reflecting the confidence levels in their conclusions. For instance, one diagnostic tool might assign a high confidence level to a cancer diagnosis based on specific symptoms and test results, while another tool, analysing the same data, might offer a lower confidence level due to certain ambiguities or conflicting indicators.

How do we effectively represent and reason with such varying degrees of uncertainty in data? Many-valued logic emerges as a powerful tool for addressing this challenge. Unlike binary logic, which offers only two truth values (true or false), many-valued logic allows for a continuum of truth values between 0 (completely false) and 1 (completely true). In the context of medical diagnosis, this means that statements like "The patient has cancer" can be assigned truth values reflecting the degree of confidence in the diagnosis.

By embracing many-valued logic, we can capture the nuances of uncertainty inherent in many of the data employed in several fields and provide more nuanced interpretations of diagnostic results. Moreover, many-valued logic allows the integration of crisp and uncertain data sources in a unified formalism.

Well-known attempts related to Datalog to handle partially true data with their limitations have been thoroughly described in sections 2.7 and 2.8.

In this section, we present extensions of Datalog and Datalog[±] to the many-valued

semantics of infinite-valued Łukasiewicz logic. We argue that these languages offer natural semantics for practical applications with uncertain data while preserving key theoretical properties of the classical (two-valued) case. Overall, the contributions of this section are summarised as follows.

- We introduce MV-Datalog as Datalog with Łukasiewicz semantics. We discuss adaptations of the standard minimal model concept to fuzzy models where every fact takes the least possible truth degree. We show that, for consistent inputs, unique minimal models always exist and give a characterisation of such minimal models. Our results also hold for K -fuzzy models, i.e., interpretations that satisfy formulas of the program only to degree K rather than fully.
- We show that MV-Datalog is suitable for practical applications. For one, our characterisation reveals that finding K -fuzzy models is feasible by building on standard methods as from the classical setting. In particular, we also show that fuzzy fact entailment in MV-Datalog has the same complexity as fact entailment in Datalog.
- We propose MV-Datalog[±] as a practice-oriented extension of MV-Datalog featuring existential quantification in the head. To obtain intuitive semantics for logic programming, we deviate from the standard semantics of many-valued existential quantification and instead argue for the use of strong existential quantification. We discuss fundamental model-theoretic properties and challenges of MV-Datalog[±] and show how our characterisation results for MV-Datalog can be extended to capture a natural class of preferred models of MV-Datalog[±], providing the foundation for a system implementation of MV-Datalog[±].

Structure This section is structured as follows. We first introduce MV-Datalog and our main results for minimal fuzzy models in MV-Datalog. Then, we proceed to present the further extension to MV-Datalog[±]. We conclude with final remarks on this work.

3.1.1 MV-Datalog and Minimal Fuzzy Models

Here, we propose the extension of Datalog to Łukasiewicz semantics for many-valued reasoning. We show that MV-Datalog allows for an analogue of minimal model semantics in the fuzzy setting. Moreover, we give a characterisation of such

minimal models in terms of an optimisation problem over the ground program corresponding to the oblivious chase.

Łukasiewicz Logic While there are many semantics for many-valued logic, in this work we will focus on infinite-valued Łukasiewicz logic $\mathbb{L}$ (see, [83]). Some of our techniques are particular to Łukasiewicz logic, and the results given in this section generally do not apply to other many-valued logics. For some relational signature σ , we consider the following logical language, where R is a relational atom (in σ) and a formula φ is defined via the grammar

$$\varphi ::= R \mid \varphi \odot \varphi \mid \varphi \oplus \varphi \mid \varphi \rightarrow \varphi \mid \neg \varphi$$

For a signature σ and domain Dom , let $GAtoms$ be the set of all ground atoms with respect to σ and Dom . A *truth assignment* is a function $v: GAtoms \rightarrow [0, 1]$, intuitively assigning a degree of truth in the real interval $[0, 1]$ to every ground atom. By slight abuse of notation, we also apply v to formulas to express the truth of ground formulas γ, γ' according to the following inductive definitions.

$$\begin{aligned} v(\neg \gamma) &= 1 - v(\gamma) \\ v(\gamma \odot \gamma') &= \max\{0, v(\gamma) + v(\gamma') - 1\} \\ v(\gamma \oplus \gamma') &= \min\{1, v(\gamma) + v(\gamma')\} \\ v(\gamma \rightarrow \gamma') &= \min\{1, 1 - v(\gamma) + v(\gamma')\} \end{aligned}$$

That is, $\odot$ is the usual Łukasiewicz *t-norm* and $\oplus$ is the corresponding *t-conorm*, which take the place of conjunction and disjunction, respectively. Implication $\gamma \rightarrow \gamma'$ could equivalently be defined as usual as $\neg \gamma \oplus \gamma'$. Note that De Morgan's rule holds as usual for $\odot$ and $\oplus$.

For rational $K \in (0, 1]$ we say that a formula φ is *K-satisfied* by v if for every grounding γ of φ over Dom it holds that $v(\gamma) \geq K$. Whenever we make use of K in this context we assume it to be rational. In the context of rule-based reasoning, it may be of particular interest to observe that an implication is 1-satisfied exactly when the head is at least as true as the body. For a set of formulas Π , we say that a truth assignment v is a *K-fuzzy model* if all formulas in Π are *K-satisfied* by v .

In place of the database in the classical setting, we instead consider (*finite*) *partial truth assignments*, that is, partial functions $\tau: GAtoms \rightarrow (0, 1]$ that are defined for a finite number of ground atoms. Let (Σ, τ) be a pair where Π is a set of formulas and τ is a partial truth assignment, a *K-fuzzy model* of (Π, τ) is a *K-fuzzy model* v of Σ where $v(G) = \tau(G)$ for every ground atom G for which τ is defined. Whenever

we talk about formulas Σ and partial truth assignments τ we use $ADom$ to refer to their *active domain*, i.e., the subset of the domain that is mentioned in either Σ or $\{G \in GAtoms \mid \tau(G) \text{ is defined.}\}$. We write $GAtoms[Adom]$ to indicate $GAtoms$ restricted to groundings over $Adom$.

K-fuzzy models and their theory have been introduced and studied in the context of logic programming by [57] but under different many-valued semantics based on the Gödel t-norm.

Example 8. Consider a labelling of images as described in the introduction, represented by predicate *Label*. Additionally, we have a predicate corresponding to whether the image was taken in a polar region. Suppose we have the following observations for image i_1 .

$$\tau(\text{Label}(i_1, \text{Whale})) = 0.8 \quad \tau(\text{PolarRegion}(i_1)) = 0.7$$

Now, we consider the satisfaction of the following ground rule under the semantics studied by [57] in contrast to Łukasiewicz semantics.

$$\text{Label}(i_1, \text{Whale}) \odot \text{PolarRegion}(i_1) \rightarrow \text{Orca}(i_1)$$

If we interpret implication and $\odot$ under the semantics proposed by [57], the body has truth 0.7 (the Gödel t-norm of 0.8 and 0.7). The implication evaluates to the maximum of 0.3 ($= 1 - \text{truth of the body}$) and the truth of the head. Hence, a model that 1-satisfies the rule would have to satisfy $\text{Orca}(i_1)$ with truth degree 1. This can be unintuitive, especially in the context of logic programming, since the truth degree of the fact is not inferred from the truth of the body.

In Łukasiewicz semantics, the body has truth degree 0.5 following the intuition that the conjunction of two uncertain observations becomes even less certain. For the implication to be 1-satisfied it suffices if $\text{Orca}(i_1)$ is true to at least degree 0.5. We see that the inferred truth degree of the head naturally depends on the truth of the body.

MV-Datalog

Definition 25 (MV-Datalog Program). An *MV-Datalog program* Σ is a set of $\mathbb{L}$ formulas of the form

$$B_1 \odot B_2 \odot \dots \odot B_n \rightarrow H$$

where all B_i , for $1 \leq i \leq n$, and H are relational atoms.

3. Reasoning In Many-Valued Settings

As noted above, we consider a partial truth assignment τ in place of a database. We will, therefore, refer also to τ as the database. We call a pair Σ, τ of an MV-Datalog program and database an *MV-instance*. We can naturally map a MV-Datalog program Σ to a Datalog program by substituting $\odot$ with $\wedge$. We refer to the resulting Datalog program as Σ^{crisp} . For the respective crisp version of τ we write D_τ for the database containing all facts G for which $\tau(G)$ is defined and greater than 0.

Analogous to fact entailment in classical Datalog, we have the central decision problem K-Truth, whether a fact is true to at least a degree c in all models, as follows.

K-Truth

Input MV-instance (Σ, τ) , ground atom G , $c \in [0, 1]$

Output $\nu(G) \geq c$ for all K -fuzzy models ν of (Σ, τ) ?

MV-Datalog is a proper extension of Datalog in that, for $K = 1$ and when all ground atoms in the database are assigned truth 1, its models coincide exactly with those of Datalog programs.

Note that, in contrast to Datalog, MV-instances do not always have models. For example, consider a program consisting only of rule $R(x) \rightarrow S(x)$. For any $K > 0$, there is a database τ that assigns truth 1 to $R(a)$ and some truth less than K to $S(a)$ such that the rule is not K -satisfied. This is a consequence of the definition of a K -fuzzy model ν of (Σ, τ) , which requires the truth values in ν to agree exactly with τ , for every fact for which τ is defined. In some settings, it may be desirable to relax this slightly and consider K -fuzzy models ν where $\nu(G) \geq \tau(G)$ where $\tau(G)$ is defined. Note that considering $\nu(G) \leq \tau(G)$ allows the trivial model to set everything to truth 0. Our semantics covers such a relaxation since it can be simulated by straightforward rewriting of the program: for every relation symbol R that occurs in τ add rule $R(\mathbf{x}) \rightarrow R'(\mathbf{x})$, and replace all other occurrences of R in the program by R' .

Ebrahim [57] introduced minimal K -fuzzy models to study logic programming with the Gödel t-norm, as the intersection of all K -fuzzy models, where truth values of an intersection are taken to the minimum between the two values. We will use the following alternative (but equivalent) definition here that will be more convenient in our setting. For two truth assignment ν, μ we write $\nu \leq \mu$ when for

all $G \in GAtoms$, it holds that $\nu(G) \leq \mu(G)$. We similarly write $\nu < \mu$ if $\nu \leq \mu$ and $\nu(G) < \mu(G)$ for at least one $G \in GAtoms$.

Definition 26. Let Σ, τ be an MV-instance. We say that a K -fuzzy model μ of (Σ, τ) is *minimal* if for every K -fuzzy model ν of (Σ, τ) it holds that $\mu \leq \nu$.

Minimal K -fuzzy models behave very similarly to minimal models in Datalog. In particular, the two following propositions state two key properties that allow us to focus on a single minimal fuzzy model for the problem K-Truth.

Theorem 1. Let Σ, τ be an MV-instance. For every rational $K \in (0, 1]$, if (Σ, τ) is K -satisfiable, then there exists a unique minimal K -fuzzy model for (Σ, τ) .

Proof of Theorem 1. We show that, for every set of K -fuzzy models S , the infimum of S , defined as the truth assignment ν' with $\nu'(G) = \inf_{\nu \in S} \nu(G)$ for all $G \in GAtoms$, is also a K -fuzzy model for (Σ, τ) .

For any ground rule γ and $\nu \in S$, the body truth value $\nu(\text{body}(\gamma))$ is given by $\max\{0, (\sum_{i=1}^{\ell} \nu(G_i)) - \ell + 1\}$ where the body of γ consists of ground atoms $G_1, \dots, G_{\ell}$. Hence, since we have $\nu'(G_i) \leq \nu(G_i)$ for every $1 \leq i \leq \ell$ by construction, it also holds that $\nu'(\text{body}(\gamma)) \leq \inf_{\nu \in S} \nu(\text{body}(\gamma))$.

Therefore also $\nu'(\text{body}(\gamma)) \leq (1 - K) + \inf_{\nu \in S} \nu(\text{head}(\gamma))$, since all $\nu \in S$ are K -fuzzy models. That is, there exists a rational $r = \max\{0, \nu'(\text{body}(\gamma)) - 1 + K\}$ (recall that we only consider rational K) such that r is less or equal to $\nu(\text{head}(\gamma))$ for every $\nu \in S$ and $\nu'(\text{body}(\gamma)) \leq (1 - K) + r$. Then, $\nu'(\text{head}(\gamma)) = \inf_{\nu \in S} \nu(\text{head}(\gamma)) \geq r$ and therefore also $\nu'(\text{body}(\gamma)) \leq (1 - K) + \nu'(\text{head}(\gamma))$, i.e., ν' K -satisfies γ . We see that ν' is indeed a K -fuzzy model, and clearly, for every $\nu \in S$, we also have $\nu' \leq \nu$. Taking the infimum over all K -fuzzy models of (Σ, τ) yields the unique minimal K -fuzzy model for (Σ, τ) . $\square$

Proposition 2. Let G be a ground atom, $c \in [0, 1]$, and μ the minimal K -fuzzy model of (Σ, τ) . Then $\mu(G) \geq c$ if and only if $\nu(G) \geq c$ for all K -fuzzy models of (Σ, τ) .

Of particular importance to our main result of this section is the fact that fuzzy models are, in a sense, only more fine-grained versions of classical models. In particular, since the classical case corresponds to every observation having maximal truth degree, fuzzy models are, in a sense, bounded from above by classical models.

Lemma 3. *Let (Σ, τ) be an MV-instance. Let ν be a minimal K -fuzzy model of (Σ, τ) . Then, for every $G \in GAtoms$, we have $\nu(G) > 0$ only if $\Sigma^{crisp}, D_\tau \models G$.*

Proof of Lemma 3. Let G be a ground atom with $\nu(G) > 0$. Since ν is minimal, there must be a ground rule with G in the head such that for every body atom $G_i \in \text{body}(\gamma)$ also $\nu(G_i) > 0$. Consider the tree with γ as the root and a child for every body atom of γ . At each child, take again a rule where the respective body atom is in the head, and the body is true to some degree greater than 0.

Repeating the process, we must arrive at a finite tree where all leaves must correspond to ground rules where all body atoms are defined in τ with truth greater than 0. Otherwise, we could simply set $\nu(G') = 0$ for all ground atoms G' whose branches do not lead to such leaves and ν would still be a K -fuzzy model, contradicting its minimality.

Viewing the same tree in the context of Σ^{crisp}, D_τ , we see, going from the leaves upward, that the bodies of all these rules must be true in all models. Thus, also ultimately $\Sigma^{crisp}, D_\tau \models G$. $\square$

Characterising Minimal Fuzzy Models

Minimal K -fuzzy models are a natural tool for query answering. In the following, we show that we can, in fact, characterise the minimal K -fuzzy model for (Σ, τ) in terms of a simple linear program $Opt_{\Sigma, \tau}^K$ defined over the ground rules induced by the oblivious chase sequence for Σ^{crisp}, D_τ . While using the oblivious chase without existential quantification is untypical, we explicitly require all rules induced by oblivious applications for our characterisation.

Let (Σ, τ) be an MV-instance. Let $\Gamma = \{\gamma_1, \dots, \gamma_m\}$ be the ground rules in $OGround(\Sigma^{crisp}, D_\tau)$. Let $\mathcal{G} = \{G_1, \dots, G_n\}$ be all of the ground atoms occurring in rules in Γ . For every $G_i \in \mathcal{G}$, we associate G_i with a variable x_i in our linear program that intuitively will represent the truth degree of G_i . For γ_j of the form $G_{j_1}, \dots, G_{j_\ell} \rightarrow G_{j_{head}}$ define

$$\text{Eval}(\gamma_j) := \sum_{k=1}^{\ell} (1 - x_{j_k}) + x_{j_{head}}$$

which directly expresses the satisfaction of rule γ_j , with variable x_{j_k} representing

the truth of G_{jk} . The linear program $Opt_{\Sigma, \tau}^K$ is then defined as follows

$$\begin{aligned}
 & \text{minimise} && \sum_{i=1}^n x_i \\
 & \text{subject to} && \text{Eval}(\gamma_j) \geq K && \text{for } 1 \leq j \leq m \\
 & && x_i = \tau(G_i) && \text{for } i \text{ where } \tau(G_i) \text{ is defined} \\
 & && 1 \geq x_i \geq 0 && \text{for } 1 \leq i \leq n
 \end{aligned} \tag{3.1}$$

Recall that a *solution* of a linear program is any assignment to the variables, a *feasible solution* is a solution that satisfies all constraints, and an *optimal solution* is a feasible solution with minimal value of the objective function. With this in mind, we can state our main result for MV-Datalog, a characterisation of minimal K -fuzzy models in terms of optimal solutions of $Opt_{\Sigma, \tau}^K$.

Theorem 4. *Let Σ, τ be an MV-instance. Then $\mathbf{x}$ is an optimal solution of $Opt_{\Sigma, \tau}^K$ if and only if $\nu_{\mathbf{x}}$ is a minimal K -fuzzy model of (Σ, τ)*

By construction, any feasible solutions of $Opt_{\Sigma, \tau}^K$ induces a K -fuzzy model $\nu_{\mathbf{x}}$ that assigns $\nu_{\mathbf{x}}(G_i) = x_i$ and 0 to all other ground atoms not in $\mathcal{G}$. Similarly, in the other direction, we can observe that any K -fuzzy model ν of (Σ, τ) can be used to construct a feasible solution of $Opt_{\Sigma, \tau}^K$. This natural correspondence of solutions of $Opt_{\Sigma, \tau}^K$ and K -fuzzy models of (Σ, τ) is formalised in the following lemma.

Lemma 5. *Let Σ, τ be an MV-instance. The following statements hold:*

1. *For any feasible solution $\mathbf{x}$ of $Opt_{\Sigma, \tau}^K$, $\nu_{\mathbf{x}}$ is a K -fuzzy model of (Σ, τ) .*
2. *For any K -fuzzy model ν of (Σ, τ) , the solution $\mathbf{x}$ with $x_i = \nu(G_i)$ is a feasible solution of $Opt_{\Sigma, \tau}^K$.*
3. *$Opt_{\Sigma, \tau}^K$ is feasible if and only if (Σ, τ) is K -satisfiable.*

Proof of Lemma 5. Let $\mathbf{x}$ be a feasible solution of $Opt_{\Sigma, \tau}^K$. Observe that $\text{Eval}(\gamma) = \nu_{\mathbf{x}}(\gamma)$ for every ground rule γ : Since, $\text{Eval}(\gamma) \geq K$ for all $\gamma \in \Gamma$, we also see that $\nu_{\mathbf{x}}$ K -satisfies every rule in $OGround(\Sigma^{crisp}, D_{\tau})$. By definition of an oblivious chase sequence and since $\nu_{\mathbf{x}}(G) = 0$ for any ground atom that is not mentioned in $OLim(\Sigma^{crisp}, D_{\tau})$, any other ground rule $\gamma \notin \Gamma$ is trivially K -satisfied since both head and body have truth 0.

Let ν be a K -fuzzy model of (Σ, τ) . Let $\mathbf{x}$ be a solution of $Opt_{\Sigma, \tau}^K$ with $x_i = \nu(G_i)$ for all $G \in \mathcal{G}$. Since ν K -satisfies all ground rules, we in particular have that

$v(\gamma) = \text{Eval}(\gamma) \geq K$ for all $\gamma \in \Gamma$ and therefore $\mathbf{x}$ is feasible. The third statement immediately follows from the combination of the first two. $\square$

We say that a ground rule $\gamma \in \Gamma$ is ν -tight if $v(\gamma) = K$, that is, $v(\text{body}(\gamma)) - 1 + K = v(\text{head}(\gamma))$. When a rule is not tight we refer to $v(\text{head}(\gamma)) - (v(\text{body}(\gamma)) - 1 + K)$ as the ν_x -gap of the rule. The gap is at least 0 for any K -satisfied ground rule. We will be interested in the structural relationships between tight rules. We refer to the ground atoms in $\mathcal{G}$ for which τ is not defined as the *derived ground atoms* of $\text{Opt}_{\Sigma, \tau}^K$.

Lemma 6. *Let ν be a minimal model of an MV-instance Σ, τ . Let $\mathcal{G}'$ be a non-empty set of ground atoms for which ν assigns truth greater than 0. Then there exists at least one ν -tight ground rule in $\text{OGround}(\Sigma^{\text{crisp}}, D_\tau)$ such that $\text{head}(\gamma) \in \mathcal{G}'$ and $\text{body}(\gamma) \cap \mathcal{G}' = \emptyset$.*

Proof of Lemma 6. By Lemma 3, and the fact that the oblivious chase sequence constructs minimal models for Datalog, we also have that $\mathcal{G}' \subseteq \text{OLim}(\Sigma^{\text{crisp}}, D_\tau)$. It follows by construction that every derived $G \in \mathcal{G}$ in the head of some ground rule in $\text{OGround}(\Sigma^{\text{crisp}}, D_\tau)$. What is left to show is that at least one such rule is also ν -tight.

Suppose the statement is false and let $\delta > 0$ be the minimal ν -gap of rules in Γ whose body atoms are all not in $\mathcal{G}'$. Let Γ' contain only the ν -tight rules in $\text{OGround}(\Sigma^{\text{crisp}}, D_\tau)$ that contain an atom from $\mathcal{G}'$ in the head. Since we assume the statement false, all the rules in Γ' have a body atom that is also in $\mathcal{G}'$. Now consider the truth assignment ν' defined as

$$\nu'(G) = \begin{cases} \max\{0, \nu(G) - \delta\} & \text{if } G \in \mathcal{G}' \\ \nu_x(G) & \text{otherwise} \end{cases}$$

By assumption, all ν -tight rules where $\mathcal{G}'$ occurs in the head also have some atom from $\mathcal{G}'$ in the body. Meaning that under ν' , the truth of both head and body is decreased by at least δ (or until both are 0). Hence, all rules in Γ' are still K -satisfied. We can also see that all other rules that contain an atom in $\mathcal{G}'$ remain K -satisfied under ν' . When an atom of $\mathcal{G}'$ occurs in the body of a rule, the implication can only become more true. Where an atom G from $\mathcal{G}'$ occurs only in the head of a rule $\gamma \notin \Gamma'$, we have that the ν_x -gap of the rule is at least δ . That is,

$$\nu_x(G) \geq \nu_x(\text{body}(\gamma)) - 1 + K + \delta$$

and therefore also

$$v'(G) \geq v_x(G) - \delta \geq v_x(\text{body}(\gamma)) - 1 + K \geq v'(\text{body}(\gamma)) - 1 + K$$

Therefore, we observe that v' is a K -fuzzy model with $v' < v$, and we arrive at a contradiction. $\square$

Proof of Theorem 4. Observe that, by Lemma 3 and the fact that the chase produces all atoms that can be entailed by (Σ, τ) , it follows that we do not have to consider solutions with atoms other than the set $\mathcal{G}$ considered in the definition of $\text{Opt}_{\Sigma, \tau}^K$.

For the if direction, let μ be a minimal model of (Σ, τ) with $\mu(G) = 0$ for all ground atoms $G \notin \mathcal{G}$. By Lemma 5, μ induces a feasible solution $\mathbf{x}$ of $\text{Opt}_{\Sigma, \tau}^K$ where $x_i = \mu(G_i)$. Suppose this solution is not optimal and let $\mathbf{y}$ be an optimal solution of $\text{Opt}_{\Sigma, \tau}^K$. Let $\mathcal{G}' \subseteq \mathcal{G}$ be the set of ground atoms G where $v_y(G) < \mu(G)$. Since we assume that $\sum y_i < \sum x_i$, this set is not empty. By Lemma 6 there exists a μ -tight rule γ where the head is in $\mathcal{G}'$ but none of the body atoms of γ are. It follows that

$$v_y(\text{head}(\gamma)) < \mu(\text{head}(\gamma)) = \mu(\text{body}(\gamma)) - 1 + K = v_y(\text{body}(\gamma)) - 1 + K$$

Hence, v_y does not K -satisfy γ . By Lemma 5 this contradicts the assumption that $\mathbf{y}$ is a feasible solution with lower objective than $\mathbf{x}$.

For the other direction, let $\mathbf{x}$ be an optimal solution of $\text{Opt}_{\Sigma, \tau}^K$. Suppose now that v_x is not a minimal K -fuzzy model of (Σ, τ) . Hence, there exists some model v' with $v' < v_x$. Let $\mathbf{y}$ be the solution of $\text{Opt}_{\Sigma, \tau}^K$ induced by v' . Since at least one atom is less true, and none are more true in v' we have $\sum y_i < \sum x_i$, contradicting the optimality of $\mathbf{x}$ for $\text{Opt}_{\Sigma, \tau}^K$. $\square$

Since the oblivious chase (see Section 2.4) is polynomial with respect to data complexity (without existential quantification) and finding optimal solutions of linear programs is famously polynomial [93], our characterisation of minimal K -fuzzy models also directly reveals the complexity of K -Truth. Recall that for $K = 1$, PTIME-hardness is inherited from classical Datalog (see [55]).

Corollary 7. *Fix a rational $K \in (0, 1]$. K -Truth is in PTIME with respect to data complexity. Moreover, 1-Truth is PTIME-complete in data complexity.*

We briefly note why it is important to use the oblivious chase as a basis for the ground rules that make up $\text{Opt}_{\Sigma, \tau}^K$. Consider the program $P(x), Q(x) \rightarrow S(x)$ and

$R(x) \rightarrow S(x)$. In plain Datalog, $S(a)$ can be inferred simply if either $P(a)$ and $Q(a)$, or $R(a)$ are in the database. However, in the many-valued setting the truth of $S(a)$ depends on the truth of all three facts, $P(a)$, $Q(a)$, and $R(a)$ since it needs to be at least as true as $P(x) \odot Q(a)$ and at least as true $R(a)$. Technically, this means $Opt_{\Sigma, \tau}^K$ needs to consider all possible paths of inferring each ground atom, and this corresponds exactly to the oblivious chase applying rules once per homomorphism from the body into the instance.

Classical Behaviour for Certain Knowledge

We are particularly interested in settings where one wants to combine large amounts of certain knowledge, e.g., classical knowledge graphs and databases, with additional fuzzy information. Going back to our example from the introduction, we have such a scenario if we want to reason on properties of the animals in the pictures by using pre-existing biological and zoological ontologies. Ideally, we want inference on the ontologies to behave exactly as in the Datalog and only differ in behaviour when such *certain knowledge* is combined with fuzzy observations and consequences of fuzzy observations. Note that particularly in the context of ontologies, the question of fuzziness has also been studied extensively in the field of fuzzy description logics, see, e.g., [36, 157, 109]. Notably, the $\mathcal{EL}$ family of description logics, which is closely related to rule-based languages, has also been studied specifically with fuzzy semantics based on Łukasiewicz logic and shown undecidable by [35].

We show now that MV-Datalog indeed exhibits such ideal behaviour. In particular, when we consider Lemma 3 but with a database consisting only of certain knowledge in τ , i.e., ground atoms for which τ evaluates to 1, we can state the following.

Theorem 8. *Let Σ, τ be a MV-Datalog program and database. Let D^1 be the classical database containing all $G \in GAtoms$ where $\tau(G) = 1$. If Σ, τ is 1-satisfiable, then for every ground atom G , $\Sigma^{crisp}, D^1 \models G$ if and only if $\mu(G) = 1$ where μ is the minimal 1-fuzzy model of (Σ, τ) .*

Proof of Theorem 8. The implication from right to left is a special case of Lemma 3. For the other direction, we argue by induction over the oblivious chase sequence $D_0, D_1, \dots$ for Σ^{crisp}, D^1 , that if some ground atom G is in D_i , then $\mu(G) = 1$ in every 1-fuzzy model μ of (Σ, τ) . The base case $D_0 = D^1$ follows by definition of D^1 . For the induction, suppose the claim holds up to step i . In the step from D_i to D_{i+1} ,

either the ground atoms in both are the same (the oblivious application produced a ground head that was already in D_i), or there is a single new ground atom G in D_{i+1} but not in D_i . Let γ be the ground rule induced by the oblivious application in the step from i to $i + 1$. Since the rule was applicable, for all $G' \in \text{body}(\gamma)$, we have $G' \in D_i$ and, by the induction hypothesis, also $\mu(G') = 1$. Hence, $\mu(\text{body}(\gamma)) = 1$ and since γ must be 1-satisfied, we have that in every 1-fuzzy model also $\mu(G) = 1$.

Recall, that $OLim(\Sigma^{crisp}, D^1)$ is always a minimal model of Σ^{crisp}, D^1 and thus $\Sigma^{crisp}, D^1 \models G$ if and only if $G \in OLim(\Sigma^{crisp}, D^1)$. And by the above induction argument $G \in OLim(\Sigma^{crisp}, D^1)$ implies that $\mu(G) = 1$. $\square$

Beyond the conceptual importance of Theorem 8, the statement also has significant practical consequences. Our characterisation of minimal K -fuzzy models provides a clear algorithm for computing these via the oblivious chase and then solving $Opt_{\Sigma, \tau}^K$. Using the oblivious chase is important for our characterisation since different ground rules with the same head may differ in how true their body is. In classical Datalog, it is, of course, sufficient to infer the head once, which can require significantly less effort in practice. According to Theorem 8 (and by the uniqueness of minimal fuzzy models), it is unnecessary to compute the truth degree of those facts entailed by Σ^{crisp}, D^1 via $Opt_{\Sigma, \tau}^K$. An implementation can use this observation to produce fewer ground rules (i.e., not the full oblivious chase) and smaller equivalent versions of $Opt_{\Sigma, \tau}^K$ with less computational effort. We conclude that MV-Datalog is well suited for combining large amounts of certain knowledge with fuzzy observations. Not only are the semantics on certain data equivalent to those of classical Datalog, but the computation of minimal fuzzy models also exhibits desirable properties in the presence of certain data.

3.1.2 MV-Datalog[±]

Existential quantification in the head of rules provides a natural way of dealing with the common problem of incomplete or missing data by declaratively stating that certain facts must exist. Such usage is especially powerful in applications where rule-based reasoning is used for complex data analysis (see e.g., [27]). In the following, we introduce MV-Datalog[±] as an extension of MV-Datalog, focusing on providing a fuzzy reasoning language that is useful for such applications.

We propose an alternative to the commonly studied semantics for existential quantification for Łukasiewicz logic to obtain a semantics matching our intuition for such a system. We then identify certain *preferred models* that exhibit

desired conceptual and computational behaviour as the basis for reasoning in MV-Datalog[±].

Strong Existential Quantification in Łukasiewicz Semantics

The semantics of existential quantification in Łukasiewicz logic is commonly defined as $v(\exists x \varphi(x)) = \sup\{v(\varphi[x/c]) \mid c \in \text{Dom}^{|x|}\}$ where $\varphi[x/c]$ is the substitution of variables x in φ by constants c (cf. [83]).

However, with our focus set on practical applications, we propose alternative semantics for existential quantification that match the semantics of $\oplus$. To contrast the aforementioned semantics of $\exists$, we refer to our semantics as *strong existential quantification*.

$$v(\exists x \varphi(x)) = \min\{1, \sum_{c \in \text{Dom}^{|x|}} v(\varphi[x/c])\}$$

We refer to Ł extended with strong existential quantification as Ł_∃ (following standard syntax of first-order existential quantification). Such semantics for quantification in Łukasiewicz logic have also recently been studied more generally in a purely logical context by [64].

Definition 27. A MV-Datalog[±] program is a set of Ł_∃ formulas that are either MV-Datalog rules or of the form

$$B_1 \odot B_2 \odot \dots \odot B_n \rightarrow \exists x \varphi(x, y)$$

where B_i are relational atoms, y are the variables in the body, and φ is a formula that contains only a relational atom that uses all variables of x .

Note that we generally do not restrict the domain in the semantics of Ł formulas. Thus, as in Datalog[±], interpretations are not constrained to the active domain of Σ and τ but allow for introducing new constants. As noted above, introducing new constants provides a natural mechanism to deductively handle missing and incomplete data. Our proposed use of strong existential quantification is motivated by its behaviour in cases of incomplete information as illustrated in the following example.

Example 9. We consider the following example rule, expressing that every company has a key person, to illustrate the difference between the two semantics for $\exists$.

$$\text{Company}(y) \rightarrow \exists x \text{ KeyPerson}(x, y)$$

Suppose we have the following database. We know for certain that Acme is a company and we are 0.8 degrees confident that Amy is the key person of Acme.

$$\tau(\text{KeyPerson}(\text{Amy}, \text{Acme})) = 0.8, \quad \tau(\text{Company}(\text{Acme})) = 1$$

We discuss the case in a 1-fuzzy model in the following. With existential semantics via the supremum of matching ground atoms, the first rule implies the existence of some new $\text{KeyPerson}(N_1, \text{Acme})$ with truth degree 1. With strong existential quantification, the first rule implies only truth degree 0.2 for some $\text{KeyPerson}(N_1, \text{Acme})$ since the known observation $\text{KeyPerson}(\text{Amy}, \text{Acme})$ already contributes 0.8 degrees of truth to the head.

Intuitively, the new constant N_1 assumes the role of the unknown other object that possibly is the key person. In traditional semantics for $\exists$ we have to infer (in a 1-fuzzy model) that N_1 is certain to control Acme. In contrast, under strong existential semantics, the confidence in N_1 being the key person is determined by the known observation that Amy might be the key person.

Preferred Models for MV-Datalog[±]

When considered in full generality, our proposed strong existential quantification semantics reveals various theoretical challenges. Allowing for K -fuzzy models ν where $\{G \in \text{GAtoms} \mid \nu(G) > 0\}$ is not finite would require the introduction of further technical complexity in the definition. Furthermore, it can be necessary for a model to introduce multiple new constants per ground rule to satisfy an existential quantification in certain situations, as illustrated in the following example.

Example 10. Consider database τ with $\tau(S(a)) = 0.8, \tau(T(a)) = 0.2$ and program Σ

$$\begin{aligned} S(x) &\rightarrow \exists y P(x, y) \\ P(x, y) &\rightarrow T(x) \end{aligned}$$

Then, there is no 1-fuzzy model with only one new constant. However, there is a solution where the facts $P(a, n_1), P(a, n_2), P(a, n_3), P(a, n_4)$, with new constants $n_1, \dots, n_4$, are all assigned truth degree 0.2.

Infinite models and the phenomenon illustrated by Example 10 present interesting topics for further theoretical study. For practical applications we propose to avoid these corner cases and focus on natural *preferred models* that avoid these issues and provide predictable outcomes.

Recall that the oblivious chase sequence introduces new constants exactly once for every possible true grounding of the body. This limitation on the number of new constants provides a natural balance between not overly restricting facts with null values in models while also avoiding extreme situations as in Example 10 where arbitrarily many constants are introduced to satisfy the rule for a single ground body. We also restrict the minimality of fuzzy models to only those ground atoms without labelled nulls. Minimising also the truth of atoms with labelled nulls would intuitively correspond to a kind of soft closed-world assumption, where we want to infer the truth of statements under the assumption that the unknown information is minimal.

Definition 28. Let Σ, τ be an $MV^\pm$ -instance. Let μ, ν be two K -fuzzy models of (Σ, τ) . We say that $\mu \leq_a \nu$ if for all $G \in GAtoms[Adom]$ it holds that $\mu(G) \leq \nu(G)$.

Definition 29. We say that a K -fuzzy model ν of MV -Datalog $^\pm$ instance (Σ, τ) has an *oblivious base* if $OLim(\Sigma^{crisp}, D_\tau)$ is finite, if for every $G \in GAtoms$ it holds that $\nu(G) > 0$ only if $G \in OLim(\Sigma^{crisp}, D_\tau)$. We say that ν is a *preferred K -fuzzy model* if it is minimal with respect to the partial ordering $\leq_a$ of all K -fuzzy models with an oblivious base.

This definition of preferred models captures the previously outlined intuition of which kinds of models we want to consider for practical applications. That is, we propose preferred models as the desired output of an MV -Datalog $^\pm$ system. An $MV^\pm$ -instance does not necessarily have a unique preferred model. The question of how to compare preferred models with each other is left as an open question.

We now extend the definition of $Opt_{\Sigma, \tau}^K$ from above to program $\exists\text{-}Opt_{\Sigma, \tau}^K$ for an $MV^\pm$ -instance Σ, τ . Let $\mathcal{G}$ and Γ be as in the previous definition of $Opt_{\Sigma, \tau}^K$. For ground rules that are the result of grounding a rule without existential quantification, $Eval$ is defined the same as it was for $Opt_{\Sigma, \tau}^K$. Otherwise, let γ be a ground rule obtained from the grounding of an existential rule. Let G_h be the atom in the head of γ and let N be the labelled nulls at the positions in G_h that were existentially quantified in the original rule before grounding. We say that a ground atom G *matches* G_h if G can be obtained from G_h by replacing the labelled nulls in N with some other values of the domain. Let $G_{\beta_1}, \dots, G_{\beta_k}$ be the ground atoms in $\mathcal{G}$ that match G_h . We can then extend our previous definition of $Eval$ to groundings of existential rules as

$$Eval(\gamma) := \sum_{j=1}^{\ell} (1 - x_{\alpha_j}) + \sum_{i=1}^k x_{\beta_i}$$

with variables x_{α_j} and x_{β_i} corresponding, respectively, to the truth values of the ground atoms in the body and the head. Then $\exists\text{-Opt}_{\Sigma,\tau}^K$ is the linear program defined the same as $\text{Opt}_{\Sigma,\tau}^K$, but with objective function

$$\text{minimise } \sum_{i=1}^n w_i x_i \quad (3.2)$$

where x_i provides the truth value of G_i , $w_i = 1$ if $G_i \in \text{GAtoms}[\text{Adom}]$ and $w_i = 0$ otherwise. That is, ground atoms that contain nulls do not contribute to the objective. Recall from Example 10 that this does not imply that they can be simply set to be fully true.

Theorem 9. *Let Σ, τ be a MV-Datalog program and database where $\text{OLim}(\Sigma^{\text{crisp}}, D_\tau)$ is finite. The following two statements hold.*

1. *If (Σ, τ) has a preferred K -fuzzy model, then $\exists\text{-Opt}_{\Sigma,\tau}^K$ is feasible.*
2. *For any optimal solution $\mathbf{x}$ of $\exists\text{-Opt}_{\Sigma,\tau}^K$, we have that $v_{\mathbf{x}}$ is a preferred K -fuzzy model for (Σ, τ) .*

From our definition of preferred models, we are naturally interested in identifying cases where the oblivious chase is finite. There exist syntactic restrictions for which we can show that this is always the case. In particular, we can identify programs where the oblivious chase is finite based on the notion of weakly acyclic programs as studied by [61]. Technical details of this question are discussed in the following paragraph.

Finiteness in the Oblivious Chase

Since we are particularly interested in instances where the chase is finite, it is of interest to identify fragments where this is always the case. The most prominent condition for which the finiteness of the chase in Datalog[±] is studied is the restriction to *weakly acyclic programs* 9 as first introduced by [61].

However, the standard definition of weakly acyclic is particular to the restricted chase, which does not fit our setting. This becomes apparent when considering the simple (weakly acyclic) program

$$P(x) \rightarrow \exists y P(y).$$

The oblivious chase is infinite in this case since every new instantiation of y generates a new homomorphism that satisfies the body. Since x does not occur in

the head at all, the dependency graph used in the standard definition of weakly-acyclic does not have any edges and is thus weakly acyclic.

Let the *variable expansion* $ve(\Sigma)$ of Σ be the program obtained by the following rewriting. Without loss of generality every existential rule is of the form $\varphi(\mathbf{x}) \rightarrow \exists y R(y, \mathbf{x}')$, where φ is the body formula, R is a relation symbol and $\mathbf{x}'$ are those (free) variables that occur in both the body and the head. Let $\mathbf{x}^*$ be the free variables that only occur in φ but not in the head. Then, for every such rule, we replace it with the two rules

$$\begin{aligned} \varphi(\mathbf{x}) &\rightarrow \exists y R^*(y, \mathbf{x}', \mathbf{x}^*) \\ R^*(y, \mathbf{x}', \mathbf{x}^*) &\rightarrow R(y, \mathbf{x}') \end{aligned}$$

to obtain $ve(\Sigma)$. Intuitively, the variable expansion reveals those particular cycles that are harmless in the restricted chase but lead to infinite sequences in the oblivious chase in the standard definition of weak acyclicity. For example, the variable expansion of our simple example above would thus be the (no longer weakly acyclic) program

$$\begin{aligned} P(x) &\rightarrow \exists y P^*(y, x) \\ P^*(y, x) &\rightarrow P(y) \end{aligned}$$

The following theorem then follows by a similar argument as originally given for the restricted chase [61, Theorem 3.9] by additionally observing that nulls can never propagate to a position in the dependency graph with lower rank, where the rank of a position is the maximal number of special edges on an incoming path. One can then inductively bound the number of possible groundings of the body to consequently bound the number of nulls generated in the head for each position.

Theorem 10. *Let Σ, τ be a $MV^\pm$ -instance program. If $ve(\Sigma)$ is weakly acyclic, then $OLim(\Sigma^{crisp}, D_\tau)$ is finite and of polynomial size in data complexity.*

3.2 Family Link Detection in Uncertain Settings

In this section, we show a system taking advantage of the MV-Datalog[±] formalism [32]. The system is used for inferring family links in company ownership graphs. It combines fuzzy observations of possible family links from anagraphic data leveraging MV-Datalog[±]. With this approach, we achieve levels of flexibility and explainability benefits that are difficult to match by the current Machine Learning methods. Here, we describe our approach’s key challenges and main components and experimentally illustrate its effectiveness at this early stage in its development.

3.2.1 Problem Introduction

Company ownership graphs play a central role for central banks, financial authorities and national statistical offices [74, 141, 45, 22] and represent a valuable tool for solving a wide range of problems in the context of banking supervision, credit-worthiness evaluation, anti-money laundering, insurance fraud detection, economic and statistical research. One particular ownership graph that we consider in this paper is the *Enterprise Knowledge Graph* of Italian companies [16], in which people and companies are the nodes, while the (labelled) edges represent the fraction of company shares owned by a person or a company.

One of the main challenges for detecting regulatory issues in the Enterprise Knowledge Graph is the obfuscation of actual company ownership. Such obfuscation can be highly problematic. For example, according to European Central Bank regulations, company C_y is not eligible as a guarantor for C_x if it is too “close” to it in terms of ownership [3]. It is therefore important to identify potential *close links* in company ownership graphs to effectively detect problematic constructs. One particularly common type of potential close link is through family ties. Oftentimes, company ownership is distributed over family members to obfuscate the actual ownership structure, leading to issues as in the example illustrated in Figure 3.1.

Consequently, if we detect a family link between a person P_a and a person P_b that respectively own a company C_x and C_y , we could infer that even if C_x and C_y do not strictly fulfil the definition of close link, we should prevent both companies from being the guarantor of the other. However, company ownership graphs are often derived from data where family relationships are not explicitly available and must be reconstructed through other methods. The lack of public data makes the task of finding family links highly challenging since civil registers can not share personal data due to privacy regulations. Note that this lack of ground truth data

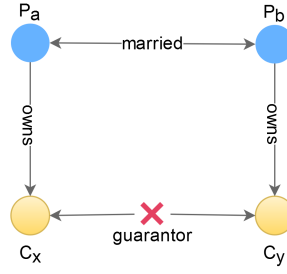


Figure 3.1: A typical case of hidden close link where C_x cannot act as guarantor for C_y due to the family relationship between P_a and P_b

also severely limits the applicability of popular Machine Learning techniques for related problems like Knowledge Graph embedding.

The scope of our analysis is directed at finding family connections. The disclosure of these relationships would greatly impact the analysis of the network by enabling the discovery of hidden links among persons and, as a consequence, companies. For instance, this could weigh on the identification of *close links* [3]. The close link problem assumes a relevant significance in the application of company graphs and consists of estimating the risk of granting a specific loan to a company C_x that is backed by collateral issued by another company C_y . According to European Central Bank regulations, company C_y is not eligible as a guarantor for C_x if it is too “close” to it in terms of ownership.

In this section, we present a novel approach to combine similarity measures that indicate potential family links with new methods for many-valued reasoning over uncertain data in KGs to obtain a unified picture of potential *family links* in the Enterprise Knowledge Graph of Italian companies.

In the first step, we infer connections between persons based on the information pertaining to company ownership and the available anagraphic data, such as addresses and names of company owners. For example, given persons P_a and P_b with the same surname, both having shares in the same companies and living in the same region, one could argue in favour of the existence of a family link between them. This hypothesis is strengthened by the knowledge of the Italian economic system, where a large number of companies have families as their major shareholders. We extend such inference beyond exact matches of data, e.g., if P_a and P_b in the scenario above lived in neighbouring regions rather than the same region the inference could still be argued, although with less certainty.

At the same time, family links observe several common logical properties, such as transitivity or symmetry, that can be used to detect further family links through logical reasoning. However, since the similarity metrics produce uncertain observations, the use of classical logical reasoning (which is limited to statements that are either fully true or false) is severely limited in this setting. Rather than reducing our uncertain observations to true/false via thresholds or similar techniques, we make use of MV-Datalog[±]. Many-valued here refers to the use of *degrees of truth*, which we use to model uncertainty, rather than the classical paradigm of every fact being either absolutely true or false. MV-Datalog[±] allows us to combine uncertain observations with logical reasoning and certain knowledge, such as a company ownership KG, expert knowledge, or known relationships, in a single framework.

In the following, we discuss how these two components can interact to provide a powerful reasoning system for detecting family links in the Enterprise Knowledge Graph for Italian companies. While the presentation focuses on this specific setting, we believe that our methods can be widely applicable and that uncertain reasoning systems open up a variety of new opportunities for the use of KGs. The rest of this section contains details of the problem setting and the derived observations of possible links between persons, a discussion on the interaction with MV-Datalog[±] for unifying and extending uncertain observations and results of experiments. Finally, the section is closed with a summary.

3.2.2 The Enterprise Knowledge Graph of Italian Companies

We use the Enterprise Knowledge Graph of Italian companies, containing the most updated data at our disposal. We focus on *non-listed companies*, i.e. public companies that are not listed on any stock exchange. For each company, the graph contains several features including legal name, registered office address, incorporation date, legal form, shareholders. A shareholder can be either a company or a person. For each person, the graph contains the standard anagraphic information. Shares can be associated with different legal rights (e.g., ownership, bare ownership, etc.). We focus on all forms of ownership and consider companies having at least one shareholder. The graph counts 11.97M nodes representing the shareholders, and 14.18M edges denoting share ownership.

3.2.3 Extracting Uncertain Family Links

In this section, we describe our technique for extracting uncertain family links from the Italian companies KG. Considering the complex analysis that we aim to perform, taking into account all the person nodes in the graph would lead to the intractability of the uncertain reasoning inference. Furthermore, most of the nodes would not be relevant to the scope of our analysis. Thus, spending computational time over those would not be wise. The first objective is to reduce the search space in such a way that it is likely a family connection exists each time two person nodes are compared.

Our approach is based on the assumption that people owning similar shares across the same companies tend to have family connections. According to these remarks, we consider the following three-step process in order to extract family links having a degree of uncertainty. The first two steps aim at reducing the search space through partitioning techniques. In the last step, each pair of nodes in the reduced space is compared to evaluate three different similarity scores. The higher the scores, the more likely the family connection.

(i) The first step involves the computation of weakly connected components (WCCs), i.e. subgraphs of the original graph where all the vertices are connected to each other by some path, ignoring the direction of edges. To compute the WCCs we exploit a more complex concept of ownership called *integrated ownership*. This notion of accumulated ownership from a company C_x to a company C_y accounts for the ownership that C_x retains of C_y along direct and indirect connections. In terms of the flow of dividends, integrated ownership can be seen as the cumulative flow from C_x to C_y , justified by direct and indirect shareholding [16].

The graph consists of about 21 million connections based on the integrated ownership. This amount of relations takes into account both direct and indirect links. The number of connections is significantly cut by removing the ones characterized by a low integrated ownership value $O < 0.001$. At this point, some of the nodes are no longer linked to any other node in the graph due to the removal of related edges.

This diminished graph presents about 1,3 million WCCs, on average composed of a number of nodes between 3 and 10, see Figure 3.2, while the largest WCC has more than 1,5 million nodes. All the WCCs composed of a single node have not been further considered. According to the WCC properties, a node is only reachable from the nodes within the same component. This feature fully satisfies

our previous assumption, since two persons could have a family link only if at least one connection (direct or indirect) through their companies exists.

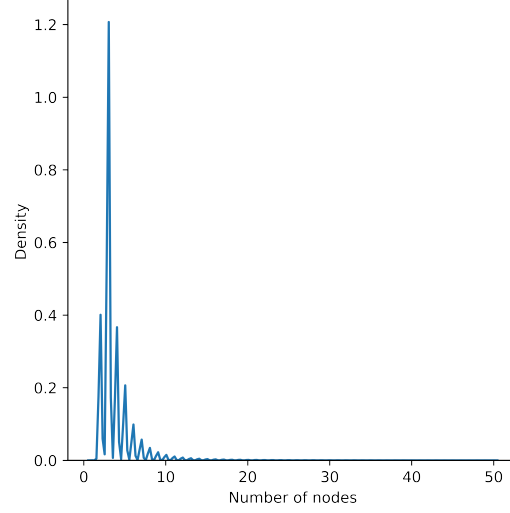


Figure 3.2: Density distribution of nodes in the WCCs of the Enterprise Knowledge Graph of Italian Companies. For the sake of visualization the communities with the higher number of nodes and density next to 0 have been omitted

(ii) In the second step, new partitions are obtained by clustering the persons having identical surnames in the same WCC. The largest WCC contains almost 60.000 clusters.

Figure 3.3 illustrates an example of the resulting clusters. The bubble colour identifies the WCC. Note that the same surname could result in more than one cluster, where each cluster belongs to a distinct WCC. For instance, there are 1500 Rossi clusters, each one contained in a different WCC. Note that this mechanism allows for finding parent-child or sibling relationships only. However, this technique could be extended to cover other common cases, for example, employing address-based clustering or considering spouse surnames.

(iii) In the third step, all the pairs of nodes possibly having a family connection, i.e. all the nodes within the same cluster, are compared to evaluate the similarity scores. The final set of candidates is composed of approximately 41 million pairs.

We compute three similarity scores on different attributes of the candidate persons. The first similarity score is based on the number of shares across the same companies. This is computed as follows:

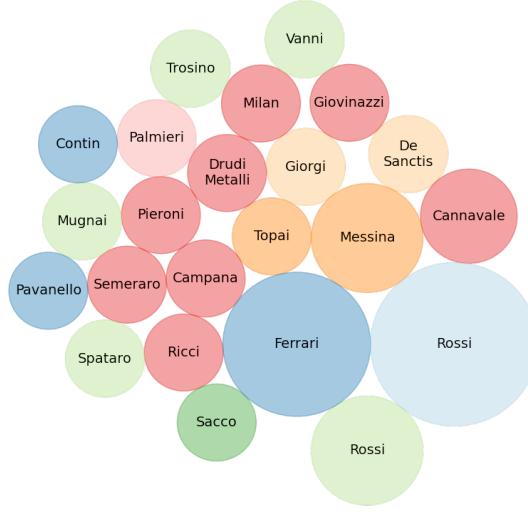


Figure 3.3: An insight of the clustering result

$$own_{P_a P_b} = \frac{\sum_{C_i \in (C_a \cap C_b)} O_{P_{a_i}} + O_{P_{b_i}}}{\sum_{C_i \in C_a} O_{P_{a_i}} + \sum_{C_i \in C_b} O_{P_{b_i}}} \quad (3.3)$$

where C_a and C_b represent, respectively, the set of companies where a person P_a and a person P_b have a percentage of integrated ownership; $O_{P_{x_i}}$ is the share of a person P_x over a company C_i . In detail, $own_{P_a P_b}$ represents the ratio between the sum of shares owned by P_a and P_b over a set of common companies and the sum of shares owned over all their companies.

Let us consider an example where three persons P_1 , P_2 and P_3 have the following shares over the companies C_x , C_y and C_z :

$$\begin{aligned} O_{P_{a_x}} &= 0.3 & O_{P_{a_y}} &= 0.2 \\ O_{P_{b_x}} &= 0.4 \\ O_{P_{c_x}} &= 0.3 & O_{P_{c_z}} &= 0.8 \end{aligned}$$

The resulting *own* scores are:

$$\begin{aligned} own_{P_a P_b} &= 0.77 \\ own_{P_a P_c} &= 0.37 \\ own_{P_b P_c} &= 0.46 \end{aligned}$$

We can notice that the score increases when there is a concentration of interest in the same companies. Indeed, while P_a and P_b ownership revolves around C_x , P_c has a relevant share of another company, C_z . This brings to lower scores associated with P_b, P_c and, in particular, P_a, P_c .

The second score takes into account the string similarity between the living addresses. Given two persons P_a and P_b having address $address_{P_a}$ and $address_{P_b}$, the similarity score is given by the Levenshtein distance [106] as follows:

$$addr_{P_a P_b} = levenshtein(address_{P_a}, address_{P_b}) \quad (3.4)$$

Finally, the third score evaluates the proximity of the address postal code, derived from the living address attribute, as:

$$postal_{P_a P_b} = \frac{k}{n} \quad (3.5)$$

Where k represents the number of the first common digits shared by the two postal codes and n is the postal code length. For instance, the postal codes 00100 and 00198 have the first three digits in common ($k = 3, n = 5$) and the resulting similarity is 0.6.

The result of the process is summarized in the following table:

id_{P_1}	id_{P_2}	...	$own_{P_1 P_2}$	$addr_{P_1 P_2}$	$postal_{P_1 P_2}$
a	b	...	0.77	0.6	0.7
a	c	...	0.37	0.5	0.8
b	c	...	0.46	0.1	0

Table 3.1: Similarity scores table

3.2.4 Uncertain Reasoning with MV-Datalog[±]

In the application presented in this section, we use MV-Datalog[±] to reason over the uncertain observations previously described. Here, we highlight some key ways in which natural constructs in MV-Datalog[±] implement important behaviour in our family link inference system. Note that the following examples have been simplified for presentation purposes and that a full system can consider much more complex constructs.

Combining Multiple Sources of Uncertainty As previously discussed, we start from some different vague observations on similarities between persons. Ultimately, we want to combine these observations into a single judgement of how certain we are regarding the eventual family link between two people. By using MV-Datalog[±] we express how to combine these different sources of vague observations simply through the logical relationships of the observations. Note that in this case these connections are direct representations of expert knowledge but can also be derived from other sources, e.g., laws and regulations. For example, from our hypothesis, we believe that two people who own the same amount of shares in the same company (the *own* score mentioned before interpreted as a fuzzy predicate), and who live in the same region are related. Using MV-Datalog[±] we can express this directly through the simple rule for the vague observations *own* and *postal* as described above.

$$related(P_a, P_b) \leftarrow own(P_a, P_b), postal(P_a, P_b).$$

Importantly, if the region and ownership structure of two people are exactly the same, i.e., the respective *own* and *postal* observations are completely true (i.e. they hold truth degree 1), then the relatedness of the respective persons will be inferred as completely true. If one or both of the observations are uncertain, the truth degree of $related(P_a, P_b)$ is derived through more complex means in a consistent way with all the other rules.

In a similar fashion, we infer parent/child relationships for persons living at the same address where one person is much younger than the other. Again, rule-based reasoning makes it simple to specify such a rule, but also to express that the child relationship is a special case of *related*:

$$\begin{aligned} child(P_a, P_b) &\leftarrow age_gap(P_a, P_b), addr(P_a, P_b). \\ related(P_a, P_b) &\leftarrow child(P_a, P_b). \end{aligned}$$

This also illustrates how the vague observation *addr* can imply relatedness in a significantly different way than *own* or *postal*. The final solution model for the *related* relation is then derived from all these logical relationships and interactions globally through the program to provide a unified answer that is consistent with the input observations and the specified logical rules. Furthermore, it is possible to specify *weights* for rules to express their importance relative to one another and can further affect the solution of MV-Datalog[±] inference.

Fuzzy Closure over Logical Properties Family relations follow some logical axioms such as symmetry or transitivity. Naturally, if we detect that persons P_a and

P_b are related, as well as P_a and P_c , we can deduce that also P_b and P_c are related. In rule-based reasoning, all such implied connections can be easily deduced by expressing the respective properties in logical terms. With uncertain information such situations become more complex. Say we deduce $related(P_a, P_b)$ with truth 0.7 and, independently, $related(P_b, P_a)$ with truth 0.9, then it can be unclear how to consolidate this when assuming that $related$ is symmetric. Enforcing such properties via ad-hoc computation is often highly challenging especially when trying to reconcile multiple properties at the same time (such as symmetry and transitivity). MV-Datalog[±] offers natural semantics for such situations in which it will produce the equilibrium where every rule is as satisfied as possible. This allows us to express these important properties in simply their natural logical form as we would in any Datalog program for certain knowledge:

$$\begin{aligned} related(P_a, P_b) &\leftarrow related(P_b, P_a). \\ related(P_a, P_c) &\leftarrow related(P_a, P_b), related(P_b, P_c). \end{aligned}$$

Explainability and Transparency In a regulatory context, it is critical to know why a system infers family links. Many popular Machine Learning methods struggle to provide such explainability and transparency at the required level. By inferring family links through logical reasoning, our system is fully transparent and can provide full explanations, consisting of the individual steps of deduction that ultimately led to a link being inferred.

Consider a link between A and B that has been inferred with truth degree (i.e., level of certainty) 0.78 as a simple example with explanation (in formal terms):

$$\begin{aligned} child(A, C) &\leftarrow age_gap(A, C), addr(A, C) \\ related(A, C) &\leftarrow child(A, C) \\ related(C, B) &\leftarrow own(C, B), postal(C, B) \\ related(A, B) &\leftarrow related(A, C), related(C, B) \end{aligned}$$

where the observation $postal(C, B)$ has truth degree 0.8, and $addr(A, C)$ is 0.98 true, and all other input observations are certain (truth degree 1). A natural reading of the explanation is that we deduce (with certainty 0.98) that A is a child of C , making them related with the same certainty. In this simple example, C and B have the exact same company ownership and live in very similar regions; hence, we deduce they are related with truth degree 0.8. By transitivity of relatedness, this then also implies that A and B must be related and by the semantics of MV-Datalog[±]¹ the resulting fact $related(A, B)$ has truth 0.78. Note that the truth

¹In this simple example this works out to the Łukasiewicz t-norm $Truth(\alpha \odot \beta) = \max\{0, Truth(\alpha) + Truth(\beta) - 1\}$ of the two body atoms α and β .

degree is generally not derived by simple forward propagation but requires more complex considerations since different paths of rule application can lead to the same consequences (say A and B also had some common ownership). For a formal definition of the semantics of MV-Datalog[±] we refer to [101].

Advanced Data Cleaning MV-Datalog[±] also allows for *constraints*, that is, rules with *falsum* $\perp$ in their head, which express that the body of the rule shall not be satisfied. In a many-valued context, such constraints can be a valuable tool for data cleaning according to the logical properties of the domain. For example, if we are worried about data inaccuracies leading to two persons being inferred as children of each other, we can express that this is impossible through the following constraint:

$$\perp \leftarrow \text{child}(P_a, P_b), \text{child}(P_b, P_a)$$

Constraints semantically behave like all other rules in our many-valued setting in that the solution model is the model that globally satisfies all rules as much as possible, which implies violating all constraints as little as possible. Again, weighting also allows us to specify the relative importance of each constraint.

Interaction with Classical Knowledge Bases As a final point, since MV-Datalog[±] is an extension of plain Datalog, it is also well suited for tasks where we want to combine reasoning over vague inputs with classical knowledge bases. It would, for example, be natural to extend our system with various kinds of social graphs. Interaction with such classical (non-vague) knowledge bases requires no special interfaces since MV-Datalog[±] behaves exactly like plain Datalog on rules where the inputs are not vague (i.e., have truth 0 or 1).

3.2.5 Experiments

Here, we briefly report on experiments with a proof-of-concept system built following the ideas described above. We compute fuzzy relations *own*, *addr*, and *postal* following the methods and clustering previously described. Additionally, we consider a (not fuzzy) relation *age_gap* to specify whether an age gap between two persons is large enough for a parent/child relationship. Using these relations as input we use an early implementation of a MV-Datalog[±] reasoner, built on top of the state-of-the-art KGMS *Vadalog* [30] as well as the mathematical optimization solver Gurobi [82], to infer a fuzzy *related* relation (including explanations for every inferred fact) from rules that implement the main ideas shown in the previous

section, including transitive and symmetric closure and multi-layered combination of different fuzzy observations².

Our experiments followed two main goals. Understanding whether the reasoning in MV-Datalog[±] is feasible in an acceptable time for our setting and how the inferred relationships match the intuition and intention of our rules.

With respect to the performance, we ran tests with a small and a medium-sized dataset, with each aforementioned relation containing 1000 or 10000 tuples, respectively. On a standard consumer laptop (8GB Memory, 1.4 GHz Quad-Core Intel Core i5), inference of the *related* relation (as well as all corresponding explanations) takes roughly 10 seconds for the small instance and about 160 seconds for the medium-sized test dataset. Note that due to the effective partitioning of the search space through the combined use of WCCs and surname clusters, these instance sizes are already representative of many real instances. Reasoning needs to be performed only within single clusters in WCCs independently of other clusters and WCCs. We can conclude that the computational effort required of our system is low enough for our setting and we expect this to still hold when further complexity is introduced

With respect to the resulting *related* relation, we infer almost 1000 tuples for the relation in the small instance (this includes the tuples resulting from transitive and symmetric closure), with varying degrees of truth. Similarly, we infer a *related* relation containing about 9000 tuples for the medium-sized instance. Manual inspection of results and respective explanations confirm that the tuples and their inferred degree of truth match the intentions of domain experts who proposed the rules, confirming the effectiveness of our proposed approach.

²Integration of additional classical knowledge bases was not part of the tested program

3.3 Summary

In the first section of this chapter, we introduce MV-Datalog and MV-Datalog[±], two many-valued rule-based reasoning languages following Łukasiewicz semantics. We provide theoretical foundations for MV-Datalog by characterising its minimal models in terms of an optimisation problem and showing that fuzzy fact entailment in MV-Datalog has the same complexity as fact entailment in Datalog. For MV-Datalog[±] we introduce alternative semantics for existential quantification in Łukasiewicz logic that better match the popular usage of $\exists$ for handling missing data. As a first starting point for the practical use of MV-Datalog[±] we introduce preferred models and show how to compute them.

In recent work [102], we also showed the feasibility of extending MV-Datalog to allow for arbitrary t-norms in rules bodies and provided an efficient fixed-point procedure.

We argue that MV-Datalog and its extensions can be effective languages for reasoning in settings with uncertain data. Therefore, developing a system based on the results given here is the natural next step to validate efficiency and effectiveness compared to other formalisms. This also entails studying how the computation of minimal or preferred models can be further optimised, ideally requiring fewer ground rules than the methods described here.

Beyond practical considerations, a variety of natural theoretical questions have been left open. Prime among them are the questions of how fuzzy models of MV-Datalog behave with stratified negation and how to handle the behaviour of MV-Datalog[±] instances with an infinite oblivious chase. The latter is particularly intriguing since truth degrees effectively decrease monotonically along derivation paths. Therefore, we suspect that there are relevant classes of problems where the oblivious chase is infinite, but instances always permit finite fuzzy models.

Moreover, there is an intriguing possibility of combining MV-Datalog semantics with the probabilistic semantics of Problog to reason about uncertainty and likelihood as two orthogonal concepts simultaneously.

In the second section of this chapter, we address the family link detection problem in the setting of the Italian companies' knowledge graph. We present a novel proof-of-concept system based on uncertain predictions of family links paired with advanced fuzzy reasoning. We use graph-theoretical techniques to reduce the search space significantly and then compute a number of fuzzy relations that may indicate family links between persons. We discuss how fuzzy reasoning in

MV-Datalog[±] provides a powerful framework for reasoning on such relations as an alternative to popular Machine Learning methods, especially in settings with little to no training data. Early experiments suggest that this approach can produce effective systems, particularly in regulatory applications where transparency and explainability are highly valuable.

The presented system design provides high flexibility in further development. Further data sources can be introduced naturally as further input relations without additional effort. Adapting and adding rules is a straightforward task that domain experts can perform at any time.

One particular promising next step is the introduction of geocoding in addition to the computation of the score similarity $addr_{P_a, P_b}$. Geocoding overcomes certain limits of traditional methods based on string similarity. Indeed, it can reflect the real proximity of addresses according to their geographical coordinates. This addition can make our inference based on location much more robust, especially when we have two geographically close addresses with a totally diverse nomenclature and in the case where addresses present severe transliteration errors.

Further planned extensions include integrating additional knowledge graphs containing certain knowledge and, in particular, social graphs. The complex integration of various sources of certain knowledge with fuzzy observations will allow us to model family links more detailedly and produce more accurate inferred *related* relations.

Finally, the presented system focuses on comparisons among people with the same surnames. Future work will aim to identify possible links with different surnames while retaining the computational benefits of clustering as far as possible.

CHAPTER 4

Optimising Logical Reasoning

A typical neuro-symbolic problem involves fetching predictions from a neural component, such as a neural network, and combining these with results from ontological reasoning tasks, like solving CQA using a symbolic reasoner. Given the widespread adoption of Datalog-based engines, these constitute a natural choice for solving ontological reasoning tasks in the context of Neuro-Symbolic systems. Nevertheless, in a scenario such as the one depicted in Figure 1.3, the ontological reasoning performed by the MV-Datalog component represents the system’s bottleneck. This is due to the performance gap between the use of logic, which requires high computational efforts, and the use of AI-based methodologies such as neural networks that, once trained, can be employed to get predictions in near real-time. This highlights the importance of optimizing query-answering performance in Datalog reasoners.

Warded Datalog[±] [77] is a relevant Datalog[±] fragment widely used in practice by a variety of benchmarks and industrial scenarios to model and solve several real-world problems [30, 15]. A notable subfragment of Warded Datalog[±] is Piece-Wise Linear $\cup$ Warded (PWLW) [31] which combines favourable theoretical properties with strong real-world problems coverage proved by empirical evidence. The piece-wise linearity condition imposes a restriction on TGDs’ recursion as follows: the body of a TGD may contain at most one atom whose predicate is mutually

recursive with a predicate in the head. In Example 11 we present an instance of query-answering over a PWLW program.

Example 11. To see an example of ontological reasoning under PWLW Datalog[±], consider the following set of TGDs Σ . A database D containing instances of the extensional predicates edb_1 , edb_2 and edb_3 is augmented with Σ in order to answer the query Q .

$$\begin{aligned}
& \text{idb}_2(x, z_1), \text{idb}_3(x, z_2) \rightarrow \exists z_3 \text{idb}_1(x, z_3, z_2) \ (\sigma_1) \\
& \text{idb}_1(x, y, z) \rightarrow \text{idb}_2(x, z) \ (\sigma_2) \\
& \text{idb}_3(x, z), \text{edb}_2(x, y) \rightarrow \text{idb}_3(y, z) \ (\sigma_3) \\
& \text{edb}_1(x, y) \rightarrow \exists z \text{idb}_1(x, y, z) \ (\sigma_4) \\
& \text{edb}_2(x, y) \rightarrow \text{idb}_2(x, y) \ (\sigma_5) \\
& \text{edb}_3(x) \rightarrow \exists z \text{idb}_3(x, z) \ (\sigma_6) \\
& q(x, y, z) \leftarrow \text{idb}_1(x, y, z) \ (Q)
\end{aligned}$$

The TGDs σ_4 , σ_5 and σ_6 derive new intensional atoms idb_1 , idb_2 and idb_3 , respectively, where the variable z in σ_4 and σ_6 is existentially quantified, causing the derivation of new labelled nulls. The TGD σ_3 exhibits a simple case of mutual recursion as idb_3 appears both in the body and in the head of the rule. The TGDs σ_2 and σ_1 showcase a complex case of recursion using the interlaced mutually recursive predicates idb_1 and idb_2 . The predicate idb_1 is a mutually recursive predicate as it appears in both the head of σ_1 and the body of σ_2 . Vice versa, idb_2 is a mutually recursive predicate as it appears in both the head of σ_2 and the body of σ_1 . The recursion of the TGDs in this Example is piece-wise linear. In fact, the recursive TGD σ_3 has at most one recursive predicate in the body, i.e., idb_3 . This also holds for the mutually recursive TGDs σ_2 and σ_1 with idb_1 and idb_2 . The query Q asks for the facts of the atom idb_1 as the answer.

It has been shown that ontological reasoning under PWLW is NLOGSPACE-complete in data complexity [31], which implies that this problem could be solved with a parallel computation on a deterministic machine. Yet, the implementation of a parallel reasoning algorithm to solve ontological reasoning under PWLW is not trivial and can hardly be inspired by the complexity proofs used to demonstrate NLOGSPACE-completeness. We argue that ad-hoc optimisations for query-answering under the

PWLW fragment can boost the adoption of Datalog-based systems and simplify their integration in the context of NeSy solutions, diminishing the gap between the performances of neural and symbolic systems.

In general, a set of TGDs can be partitioned into subsets that can be evaluated independently, called *layers*. The evaluation of TGDs in a layer depends on facts provided by the evaluation of the TGDs belonging to other layers (i.e., sublayers) or by facts already contained in D ; thus, it can only begin when the evaluation of the sublayers is over. As illustrated in Figure 4.1, the TGDs in Example 11 can be partitioned into 5 layers where their dependencies are represented by arrows. In general, two TGDs are in the same layer if they share predicates with mutual recursion or if they are non-recursive and share the same head predicate. When a layer depends on a high number of layers, mainly if it contains TGDs with multiple intensional predicates in the body (i.e., join TGDs), its evaluation might lead to memory blow-up as potentially a high number of predicates must be fully computed and materialized. Furthermore, a chain of TGD activations fired to compute an intensional atom in the body of a rule forces a given join order. On the contrary, bringing the extensional atoms close to the TGDs where they are used, reveals the chain of joins between atoms required to evaluate the TGDs and allows for case-specific optimizations. Such critical implementation aspects show that we cannot rely on the simple piece-wise linearity condition of warded TGDs to provide an efficient ontological reasoning algorithm. To this end, we investigate the existence of a “normal form” for PWLW TGDs.

In this chapter, we show that every PWLW set of TGDs Σ can be transformed into a CQ-equivalent (see Chapter 15 in [14]) set of linear TGDs Σ' . As a key feature, linear TGDs allow at most one IDB in the body and are well-suited for a space-efficient implementation. Since the notion of linearity of TGDs has been overloaded in the literature, we clarify that we here refer to the Afrati-linear definition [6]. In the evaluation of PWLW TGDs, the facts associated with IDBs are exclusively compared against the (static) EDBs, which can be preloaded and indexed into the main memory before the evaluation starts. This implies that every intermediate IDB in Σ depends on exactly one layer, and only an additional constant portion of memory can be used for its evaluation. The presence of existential quantification combined with recursion makes the linearization procedure non-trivial as it invalidates all the standard linearization techniques applicable to Datalog programs [6]. Every TGD with more than one intensional atom in its body must be rewritten into a set of equivalent TGDs, where every non-linear TGD is replaced by a set of linear ones potentially containing artificial predicates (i.e.,

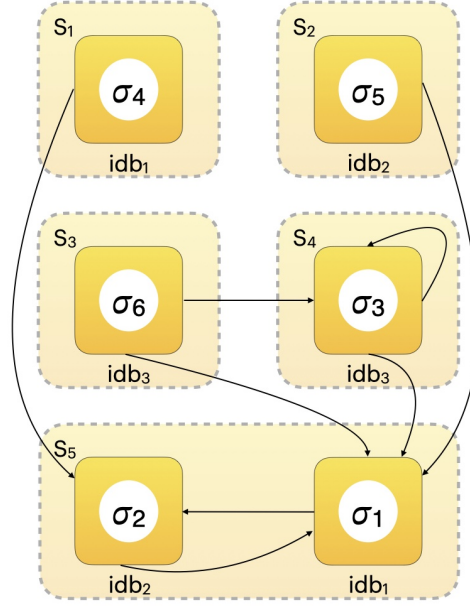


Figure 4.1: Layers dependencies for the TGDs of Σ described in Example 11.

Eureka definitions) as shown below.

Example 12. Let's consider again the set Σ from Example 11. The TGD $\sigma_1 : \text{idb}_2(x, z_1), \text{idb}_3(x, z_2) \rightarrow \exists z_3 \text{idb}_1(x, z_3, z_2)$ can be replaced by the following linear TGDs to obtain a query-equivalent linear set of TGDs.

$$\begin{aligned}
 \text{eureka}(x, x, z_2, z_1) &\rightarrow \exists z_3 \text{idb}_1(x, z_3, z_2) & (\sigma_{1,1}) \\
 \text{idb}_2(x, z_1), \text{edb}_3(y) &\rightarrow \exists z_2 \text{eureka}(x, y, z_2, z_1) & (\sigma_{1,2}) \\
 \text{edb}_2(x, y), \text{eureka}(w, x, z_2, z_1) &\rightarrow \text{eureka}(w, y, z_2, z_1) & (\sigma_{1,3})
 \end{aligned}$$

Further below, we will show that $\Sigma' = \Sigma \setminus \sigma_1 \cup \{\sigma_{1,1}, \sigma_{1,2}, \sigma_{1,3}\}$ is CQ-equivalent to Σ . The new layers dependencies for Σ' are shown in Figure 4.2

Intuitively, the linearization procedure replaces all but one intensional predicate in every non-linear TGD with all the extensional predicates on which the intensional one depends by also considering existentially quantified arguments. This turns out in iteratively creating additional TGDs and *eureka* predicates to replace an intensional predicate as shown in Example 12.

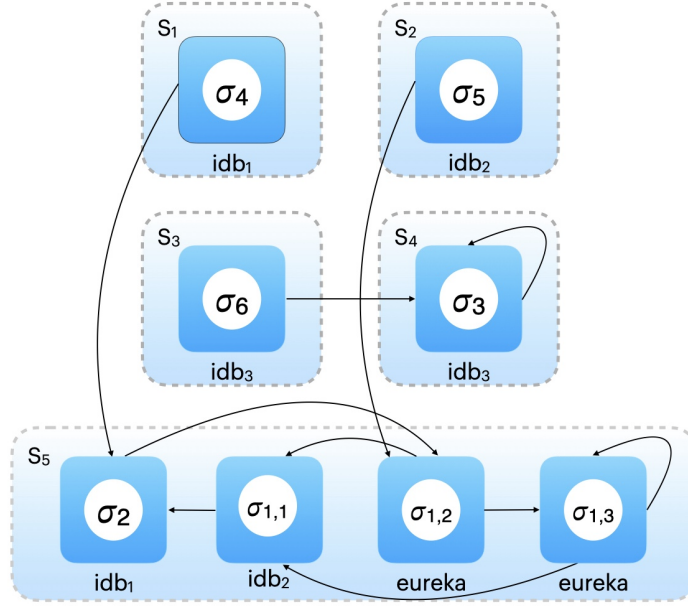


Figure 4.2: Layers dependencies for the TGDs of Σ' described in Example 12.

Our contributions can be summarized as follows:

- We prove that every set of warded TGDs with piece-wise linear recursion is linearizable, i.e., can be transformed into a CQ-equivalent set of linear TGDs.
- We provide a general linearization procedure to transform every set of warded TGDs with piece-wise linear recursion into a CQ-equivalent linear one. Such a procedure includes a novel De-Skolemization algorithm, which enables the linearization of existentially quantified rules (i.e., TGDs).
- We prove that linearizing MV-Datalog and MV-Datalog[±] programs do not change the preferred models found via $Opt_{\Sigma, \tau}^K$ and $\exists\text{-}Opt_{\Sigma, \tau}^K$. This allows us to reuse the linearization algorithm in many-valued settings without adding computational complexity.
- We describe the implementation of our linearization procedure in the Vatalog system, a state-of-the-art reasoner [25], and we show how the system exploits the linearization procedure to enable a fully-streaming and memory-bound reasoning evaluation architecture.

4.1 Linearizability of PWLW TGDs

This section is organized as follows. First, we recall the formal aspects of the PWLW fragment. Then, we present the logical toolbox and theoretical underpinnings required for the rewriting algorithm. The section continues with the introduction of a linearization algorithm for PWLW programs, along with the main theoretical results.

4.1.1 Piece-wise Linear and Warded Datalog[±]

With this work, we introduce a linearization technique for Piece-wise Linear and Warded Datalog[±] (PWLW). The PWLW fragment has been demonstrated to be NLOGSPACE-complete in data complexity which is contained in the NC₂ class of highly parallelizable problems. We argue that the key to an efficient reasoning procedure under PWLW relies on the linearization of such a fragment. Let us start laying out the fragment's definitions.

A set of TGDs Σ is PWLW if each $\sigma \in \Sigma$ respects both the piece-wise linearity (see Definition 3) and wardedness (see Definition 17) conditions. Observe that the piece-wise linearity condition can also be checked using the predicate graph $\mathcal{P}(\Sigma)$ of a set Σ of TGDs, by ensuring that every node in $\mathcal{P}(\Sigma)$ is involved in at most one cycle.

Definition 30 (PWLW TGDs). A set of TGDs Σ is PWLW if each $\sigma \in \Sigma$ respects both the piece-wise linearity and wardedness conditions.

On the other hand, a set of TGDs Σ is *Afrati-linear* if the body of every TGD σ contains at most one intensional atom. Observe that every set Σ of Afrati-linear TGDs is also PWLW as there is a single intensional predicate in the body, which is always allowed to be mutually recursive with a predicate in the head (piece-wise linearity), and the dangerous variables are forced to be propagated to the only IDB in the body (wardedness). A Warded Datalog[±] set of TGDs Σ is *harmless* if each atom in the body of each TGD shares with others only non-harmful variables (absence of harmful joins). It has been demonstrated that every set of warded TGDs can be rewritten into a CQ-equivalent Harmless Warded Datalog[±] set [25]. In the following, when mentioning PWLW sets of TGDs, we will refer to harmless PWLW sets of TGDs without loss of generality. Moreover, it is well known that TGDs with multiple head atoms can be rewritten into multiple TGDs with a single head atom in logarithmic space in a way that does not affect query answering.

Therefore, we will always refer to TGDs with a single atom in the head without loss of generality.

4.1.2 Logic Transformations

Our linearization technique is based on a logic toolbox [71, 72, 133, 136], which we adapt and extend in the context of Datalog[±]. To linearize a PWLW set of TGDs Σ , we iteratively select a non-linear TGD in Σ to be replaced by an equivalent set of linear TGDs obtained using revisited logic transformations such as fold, unfold and de-Skolemization. We recall the standard definition of these operations in the Datalog context, and then we introduce their extensions.

Unfolding without $\exists$. The operator $unfold(\sigma, \phi, \Sigma)$, where σ is a clause within the set of clauses Σ and ϕ is a predicate in σ , returns the set of clauses Σ' defined as follows. For each $\sigma'' \in \Sigma$ such that there exists a most general unifier (also "mgu", see [14] for details) θ such that $(head(\sigma''))\theta = \phi$, construct a clause $\sigma' \in \Sigma$ by replacing ϕ from σ with $(body(\sigma''))\theta$. It holds that $\Sigma = \Sigma \setminus \{\sigma\} \cup \Sigma'$.

Intuitively, the unfold operator can bring EDBs closer to the clause to linearize.

Example 13. Given the following set of Datalog rules Σ :

$$\begin{array}{ll} (\sigma_1) & idb(X, Y) \rightarrow goal(X, Y). \\ (\sigma_2) & edb(X, Y) \rightarrow idb(X, Y). \end{array}$$

The result of $unfold(\sigma, idb, \Sigma)$ is the new rule:

$$(\sigma'_1) \quad edb(X, Y) \rightarrow goal(X, Y).$$

To apply the unfold operator over a Datalog[±] set of TGDs, all atoms containing existentially quantified variables need to be *Skolemized*, i.e., replaced by Skolem functions, defined as follows.

Definition 31 (Skolemization). Let σ be a TGD of the type $\phi(\mathbf{x}) \rightarrow \exists \mathbf{z} \psi(\mathbf{y}, \mathbf{z})$, the Skolemization operation replaces $\mathbf{z}$ with a set of equality functions with Skolem terms of the form $\mathbf{z} = f(\mathbf{x})$, where $\mathbf{z} \in \mathbf{z}$. The value of each $\mathbf{z}$ is determined by the Skolem function f which is injective, deterministic and has disjunctive codomains with respect to other defined Skolem functions.

Example 14. Given the following Datalog[±] rule σ :

$$idb(X) \rightarrow goal(X, Z).$$

The Skolemization of σ is:

$$\text{idb}(X) \rightarrow \text{goal}(X, f(X)).$$

Note that terms of the form $z = f(X)$ are not allowed in Datalog rules' bodies. This is an intermediary structure that will then be simplified. Since the unfold operator is not defined over TGDs, we here introduce a procedure denominated *de-Skolemization* that we will use to overcome this limitation.

De-Skolemization. Let Σ be a PWLW set of TGDs, and $\sigma \in \Sigma$ be a TGD containing one or more Skolem terms in its body. The result of applying the De-Skolemization over σ is a (possibly empty) set of TGDs Σ' such that: (i) each TGD $\sigma' \in \Sigma'$ is Skolem terms-free; (ii) each TGD $\sigma' \in \Sigma'$ does not contain more *IDB* atoms than σ ; (iii) for any PWLW set of TGDs Σ containing σ , it holds that $\Pi = \{\Sigma \setminus \{\sigma\}\} \cup \Sigma'$ is CQ-equivalent to Σ and still in PWLW. We now present how the set of de-Skolemized PWLW TGDs Σ' is obtained. In the absence of harmful joins, we need to consider only the following cases:

- in the case of harmless joins, i.e. when the join's elements share only non-harmful variables, in the body of a TGD involving Skolem term(s), return the empty set. For example, let σ be $\phi(\mathbf{x}, y = f(\mathbf{x})), \omega(\mathbf{y}) \rightarrow \psi(\mathbf{x})$. In this case, $\Sigma' = \emptyset$ is returned since $f(\mathbf{x}) \cap \mathbf{y} = \emptyset$.
- in the case of Skolem term(s) not involved in harmless joins in the body nor present in the head of a TGD, the Skolem term(s) are replaced by a fresh variable not occurring elsewhere in Σ and the simplified TGD is returned. For example, let σ be $\phi(\mathbf{x}, y = f(\mathbf{x})), \omega(\mathbf{x}) \rightarrow \psi(\mathbf{x})$. In this case, $\Sigma' = \{\phi(\mathbf{x}, z), \omega(\mathbf{x}) \rightarrow \psi(\mathbf{x})\}$ is returned since the value for z is never used anyway.
- in the case of Skolem term(s) propagated to the head of a TGD, a level of indirection is crafted in order to remove potentially wrong dependencies at the time of reasoning. For example, let σ be $\phi(\mathbf{x}, y = f(\mathbf{x})), \omega(\mathbf{x}, \mathbf{u}, \mathbf{v}) \rightarrow \psi(\mathbf{x}, y, \mathbf{u})$. In this case, $\Sigma' = \{\sigma', \sigma''\}$ is returned, where $\sigma' = \phi(\mathbf{x}, z), \omega(\mathbf{x}, \mathbf{u}, \mathbf{v}) \rightarrow \psi'(\mathbf{x}, \mathbf{u})$ and $\sigma'' = \psi'(\mathbf{x}, \mathbf{u}) \rightarrow \psi(\mathbf{x}, y, \mathbf{u})$. When, at reasoning time, the Skolemization procedure is run on σ'' , y will be substituted with $f'(\mathbf{x}, \mathbf{u})$ which results in generating the same number of tuples for ψ . On the contrary, simply replacing $y = f(\mathbf{x})$ for z in σ , would have led to a Skolemization in which $y = f''(\mathbf{x}, \mathbf{u}, \mathbf{v})$ would have increased the number of generated tuples for ψ . Since y will never be joined on during the reasoning, focusing on producing the correct number of tuples is all that matters in this case.

We can now extend the unfold operator to the context of Datalog[±].

Unfolding with $\exists$. Let $\sigma \in \Sigma$ be an arbitrary TGD $\phi, \Omega \rightarrow \psi$, where ϕ is an atom and Ω a (possibly empty) conjunction of atoms. Let $\sigma_1, \dots, \sigma_t, \sigma_{t+1}, \dots, \sigma_m \in \Sigma$ be all the TGDs for which most general unifiers $\theta_1, \dots, \theta_m$ exist such that $\text{head}(\sigma_i)\theta_i = \phi$ for $i \in [1, m]$. Let $\sigma_{t+1}, \dots, \sigma_m$ be the TGDs that contain existentially quantified variables in the head. The result of $\text{unfold}^*(\sigma, \phi, \Sigma)$ is the set of TGDs $\Sigma' = \sigma'_1, \dots, \sigma'_m$ such that if $\sigma_j \in \Sigma$ is $\Omega_j \rightarrow \phi_j$, where Ω_j is a conjunction of atoms and $(\phi_j)\theta_j = \phi$, then σ'_j is $(\Omega_j, \Omega \rightarrow \psi)\theta_j$ for $j \in [1, m]$. For $j \in (t, m]$, i.e. in case $\text{head}(\sigma_j)$ contains an existentially quantified variable, the Skolemization procedure is run on σ_j before the unfold operator and the de-Skolemization procedure is run right after. It holds that $\Sigma = \Sigma \setminus \{\sigma\} \cup \Sigma'$.

Example 15. Given the following set of Datalog[±] rules Σ :

$$\begin{array}{ll} (\sigma_1) & \text{idb}(X, Y) \rightarrow \text{goal}(X, Y). \\ (\sigma_2) & \text{edb}(X) \rightarrow \text{idb}(X, Y). \end{array}$$

Skolemizing σ_2 :

$$(\sigma'_2) \quad \text{edb}(X) \rightarrow \text{idb}(X, f(X)).$$

The result of $\text{unfold}^*(\sigma, \text{idb}, \Sigma')$, where $\Sigma' = \Sigma \setminus \{\sigma_2\} \cup \{\sigma'_2\}$ is the new rule:

$$(\sigma'_1) \quad \text{edb}(X), Y = f(X) \rightarrow \text{goal}(X, Y).$$

Applying the de-Skolemization:

$$(\sigma''_1) \quad \text{edb}(X) \rightarrow \text{goal}(X, Y).$$

With the resulting query-equivalent program $\Sigma'' = \Sigma \setminus \{\sigma_1\} \cup \{\sigma''_1\}$

To automatically apply the unfold operator over a set of TGDs we define a mechanism to select an atom in the body to be unfolded.

Definition 32 (Unfolding Selection Rule). *An unfolding selection rule (U-rule) $\eta : \Sigma \rightarrow \text{pred}(\Sigma)$ is a partial function which maps a TGD $\sigma \in \Sigma$ to an atom $\phi \in \text{body}(\sigma)$.*

Recursively applying the unfold operator using a U-rule over the results of the unfold operator itself over a TGD wrt a set of TGDs, it is possible to construct a new data structure denominated Unfolding Tree.

Definition 33 (Unfolding Tree). Let Σ be a set of TGDs and η a U-rule. An *unfolding tree* (or U-tree) T for $\langle \Sigma, \sigma \rangle$, with $\sigma \in \Sigma$, via η is a tree labelled with TGDs, constructed as follows: (i) σ is the root label of T , and (ii) if M is a node labelled by a TGD σ and ϕ is the atom selected by η in σ , then, for each TGD σ' in $\text{unfold}^*(\sigma, \phi, \Sigma)$, there is a child node N of M labelled by σ' .

A U-tree which is constituted of only a root is called a trivial U-tree.

Definition 34 (Upper Portion of a Tree). A non-empty tree T' is called an upper portion of a tree T if (i) the root node N of T' is also the root node of T and (ii) for every node N of T' , N is also a node of T and either N is a leaf node of T' or *all* child nodes of N in T are also child nodes of N in T' .

Unfolding selecting the predicates that are non-mutually recursive with the head of a TGD usually suffices to linearize a set of TGDs, as explained in more detail later. In more complex cases, one or more of the TGDs whose head unifies with the unfolded predicate might contain a predicate mutually recursive with the head. In such cases, using the sole unfold operator would lead to non-termination. To overcome this issue, we introduce the fold operator, dual to the unfold one.

Folding without $\exists$. Let $\sigma \in \Sigma$ be an arbitrary clause $\Phi, \Omega \rightarrow \psi$, where Φ, Ω are conjunction of atoms and Ω might be empty. Let $\rho \in \Sigma$ be a clause of the form $\Phi' \rightarrow \mathbf{X}$ and let θ be a mgu such that: (i) $\Phi'\theta = \Phi$, (ii) all the variables in $\text{body}(\rho)$ which do not appear in $\text{head}(\rho)$ are mapped through θ into distinct variables which do not occur in σ and (iii) ρ is the only clause in Σ such that $\text{head}(\rho) = \mathbf{X}\theta$. The result of $\text{fold}(\sigma, \rho, \Phi, \Sigma)$ is the clause $\sigma' = \mathbf{X}\theta, \Omega \rightarrow \psi$. σ is called *folded clause*, and ρ is called the *folding clause*.

Example 16. Given the following set of Datalog rules Σ :

$$\begin{aligned} (\sigma_1) \quad & \text{edb}(\mathbf{X}, \mathbf{Z}), \text{idb}_1(\mathbf{X}, \mathbf{Y}), \text{idb}_2(\mathbf{Y}, \mathbf{Z}) \rightarrow \text{goal}(\mathbf{X}, \mathbf{Z}). \\ (\sigma_2) \quad & \text{idb}_1(\mathbf{X}, \mathbf{Y}), \text{idb}_2(\mathbf{Y}, \mathbf{Z}) \rightarrow \text{idb}_3(\mathbf{X}, \mathbf{Z}). \end{aligned}$$

The result of $\text{fold}(\sigma_1, \sigma_2, \{\text{idb}_1, \text{idb}_2\}, \Sigma)$ is the new rule:

$$(\sigma'_1) \quad \text{edb}(\mathbf{X}, \mathbf{Z}), \text{idb}_3(\mathbf{X}, \mathbf{Z}) \rightarrow \text{goal}(\mathbf{X}, \mathbf{Z}).$$

The fold operator replaces n IDB atoms in a clause with a single one, reducing the level of non-linearity, i.e. the number of IDB atoms in a clause, by a factor n at the

expense of increasing the evaluation dependencies of the clause, which require an extra evaluation step. To use the fold operator in the context of TGDs, we need to slightly modify the previous one.

Folding with $\exists$. The operator $fold^*(\sigma, \rho, \Phi, \Sigma)$, where σ is a TGD is defined as the fold operator in absence of existentially quantified variables modulo the condition (ii) which is replaced with the following: (ii') all the variables that do not appear in both $body(\rho)$ and $head(\rho)$ are mapped through θ into distinct variables which do not occur in σ .

Example 17. Given the following set of Datalog rules Σ :

$$\begin{aligned} (\sigma_1) \quad & \text{edb}(X, Z), \text{idb}_1(X, Y), \text{idb}_2(Y, Z) \rightarrow \text{goal}(X, Z). \\ (\sigma_2) \quad & \text{idb}_1(X, Y), \text{idb}_2(Y, Z) \rightarrow \text{idb}_3(X, Z, Q). \end{aligned}$$

The result of $fold(\sigma_1, \sigma_2, \{\text{idb}_1, \text{idb}_2\}, \Sigma)$ is the new rule:

$$(\sigma'_1) \quad \text{edb}(X, Z), \text{idb}_3(X, Z, Q) \rightarrow \text{goal}(X, Z).$$

4.1.3 Minimally Non-Linear TGDs

A non-trivial selection of TGDs is required to guarantee the termination of the linearization. This can be implemented by choosing the TGDs that require the slightest effort to be linearized, i.e. implementing a bottom-up approach. To define what constitutes the “bottom” in the context of the linearization procedure, we need to introduce the following definition.

Definition 35 (Predicate Transitive Closure). *The transitive closure (TC) of a predicate p wrt a set of TGDs Σ is the set $\Sigma_p \subseteq \Sigma$ inductively defined as follows: (i) if p appears in $head(\sigma)$ where $\sigma \in \Sigma$ then $\sigma \in \Sigma_p$; (ii) if $\sigma \in \Sigma_p$, then for each $p' \in body(\sigma)$, the TGDs in the transitive closure of p' wrt Σ are also in Σ_p .*

We can now define the notion of Minimally Non-linear TGDs.

Definition 36 (Minimally Non-linear TGD). Let Σ be a PWLW set of TGDs and $\sigma \in \Sigma$ a non-linear TGD. σ is Minimally Non-linear (MNL) if, for each intensional atom ϕ in $body(\sigma)$ whose predicate is not mutually recursive with the predicate in the head of σ , the TC of σ wrt Σ is Afrati-linear.

The result of unfolding MNL TGDs over the non-mutually recursive predicates in the body always leads to a linear set of TGDs. By iteratively selecting MNL TGDs, other non-linear TGDs will eventually become MNL, until all the rules are linearized.

Lemma 11. *Let Σ be a set of PWLW TGDs and $\Sigma' \subseteq \Sigma$ the set of all non-linear TGDs. Then, either Σ' is empty or there is (at least) one MNL in Σ' .*

Proof Sketch of Lemma 11. We define an ordering relation $>$, over the set Σ' consisting of all non-linear TGDs of Σ , as follows: $\sigma_1 > \sigma_2$ if σ_2 is in the transitive closure of an IDB (Definition 35) in $body(\sigma_1) \setminus \{IDB^*\}$ w.r.t. Σ , where IDB^* is the eventual IDB which is mutually recursive with the head of σ_1 . It is easy to see that $>$ is asymmetric since otherwise Σ would not be PWLW. The MNL TGDs of Σ are the minimal elements of Σ' . $\square$

Lemma 11 guarantees a starting point for the linearization procedure. Linearizing one by one all the Minimally Non-linear TGDs will eventually reduce any PWLW set of TGDs to a normal Afrati-linear form. The unfold operator is used to linearize rules: each unfolding step is determined by a linear-unfolding selection rule, which selects an intensional atom from the body of the target TGD and is defined as follows.

Definition 37 (Linear-Unfolding Selection Rule). *An unfolding selection rule η is linear (linear U-rule) if for any TGD $\sigma \in \Sigma$, η selects an intensional atom $\phi \in body(\sigma)$ such that the TC of σ wrt Σ contains only linear TGDs.*

For each MNL TGD σ , there exists a linear U-rule since σ is guaranteed to have a predicate whose transitive closure is linear.

Lemma 12. *Every TGD in a U-tree whose root is a MNL TGD unfolded via a linear U-rule η is either a linear or a MNL TGD.*

Proof Sketch of Lemma 12. By contradiction, assume that a TGD in a U-tree unfolded via the linear U-rule η returns a non-linear TGD. If that is the case, it means the atom selected for the unfolding has a transitive closure w.r.t the program which is not linear. Moreover, the de-Skolemization procedure only introduces linear TGDs, therefore we reach a contradiction. $\square$

4.1.4 Linearization Procedure

During the unfolding of a MNL clause, it is always possible to find one of two kinds of finite upper portions at some point.

Definition 38. *E/F-Linearizable Upper Portion of a Tree.* Let Σ be a Harmless PWLW program, η a U-rule, T a U-tree for $\langle \Sigma, \sigma \rangle$ via η , where $\sigma \in \Sigma$. We define a finite upper portion U of T as follows.

(1) *E-linearizable* if each leaf in U is either a linear TGD or a “Eurekable” TGD. A TGD $\sigma \in U$ is *Eurekable* if there is an ancestor $\sigma' \in U$ of σ such that, given the tuple of all the intensional atoms Φ in σ and Φ' in σ' and the mgu θ , $\Phi\theta = \Phi'$. σ' is called a folding ancestor of σ .

(2) *F-linearizable* wrt a set of brand new TGDs called *Eureka definitions* ED if each leaf in U is either a linear TGD or can be folded into a linear TGD using as folding TGD a definition in ED . The definition of ED will be treated further down in the Proof of Theorem 14.

We say that U is a *minimal E-linearizable* (resp. *F-linearizable*) *upper portion* of a U-tree T if there exists no smaller E-linearizable (resp. F-linearizable) upper portion U' of T , which is also an E-linearizable (resp. F-linearizable) upper portion of U .

Lemma 13. *Let Σ be a PWLW set of TGDs, $\sigma \in \Sigma$ be a MNL TGD, η be a linear U-rule, T be a non-trivial U-tree for $\langle \Sigma, \sigma \rangle$ via η , then, there always exists an E-linearizable upper portion of T .*

Proof Sketch of Lemma 13. Since η is a linear U-rule, it always selects atoms whose transitive closures are linear programs. Thus, unfolding a TGD σ always returns TGDs where the number of intensional atoms in their bodies is less than or equal to the number of the intensional atoms in $body(\sigma)$. Since the number of intensional predicates in Σ is finite, it follows that for every branch of T , we can find in a finite number of steps from the root of T either a linear TGD or a TGD σ' which is Eurekable with an ancestor σ'' . Therefore, there exists a finite upper portion of T with the required properties. In case T is a trivial U-tree, it means de-Skolemization removed all root’s child nodes, making the root TGD useless. $\square$

We now introduce the main theorem about the linearizability of PWLW TGDs.

Theorem 14. *For every PWLW set of TGDs Σ there exists a set of Afrati-linear TGDs Σ' such that, for every CQ Q , $D \cup \Sigma \models Q$ iff $D \cup \Sigma' \models Q$.*

Proof of Theorem 14. To show that every PWLW set of TGDs Σ can be linearized, we have to prove that each non-linear TGD $\sigma \in NLT \subseteq \Sigma$ of non-linear TGDs can be linearized or removed preserving CQ-equivalence. As stated by [27] for each set of PWLW TGDs there always exist an equivalent set of Harmless PWLW TGDs. By Lemma 11 we know that either $NLT = \emptyset$, i.e. Σ is already linear, or there is at least one MNL TGD in Σ . In the latter case, we can select a MNL TGD $\sigma \in NLT$ as the root of an unfolding tree $T = \langle \Sigma, \sigma \rangle$ obtained using a linear U-rule η . By Lemma 12, we know that each TGD in T will be either linear or MNL. Moreover, Lemma 13 guarantees the existence of an E-linearizable upper portion U of T . Each leaf l in U is either linear or Eurekaable via an ancestor $\sigma' \in U$. For each Eurekaable leaf, introduce a Eureka definition ed in a special set ED such that all the intensional atoms in both $body(\sigma')$ and $body(l)$ are instances of $body(ed)$ and $head(ed) = eureka(\mathbf{X})$, where $eureka$ is a freshly introduced predicate symbol not appearing in Σ and $\mathbf{X}$ is the minimal subset of variables in $body(ed)$ such that both l and σ' can be folded into linear TGDs using as folding TGD ed . All the leaves are either linear or folded into linear TGDs, leaving ED to be the only set of non-linear TGDs in U . Each $ed \in ED$ is MNL since is constructed using the intensional atoms of TGDs derived by the unfolding of MNL TGDs. For each $ed \in ED$, build a U-tree $T' = \langle \Sigma, ed \rangle$ using the same linear U-rule η . It is always possible to construct a non-trivial upper portion U' of T' that is F-linearizable wrt ed . To prove this, it suffices to notice that by reutilizing η , U' will be derived in a similar fashion to a subtree of T containing TGDs that led to the introduction of ed . The bodies of the TGDs in U' will have the same intensional atoms and a subset of the extensional atoms present in the TGDs in the subtree of T . It is now possible to substitute ed with the CQ-equivalent set of linear leaves of U' plus the leaves which can be folded using ed as the folding TGD. One can now substitute σ with the set of linear TGDs derived during the process as either linear leaves or folded TGDs. The process reiterates over the next MNL TGD in $NLT = NLT \setminus \sigma$ until $NLT = \emptyset$. The resulting set of TGDs is PWLW, Afrati-linear and equivalent to Σ . $\square$

Algorithm 1 shows the procedure to linearize a single TGD.

Algorithm 1: TGD Linearization Procedure (TLP)

Input : a PWLW set of TGDs Σ , a MNL TGD $\sigma \in \Sigma$ and a linear U-rule η

Output : a set LT of linear TGDs s.t. Σ is CQ-equivalent to $\Sigma \setminus \{\sigma\} \cup LT$

- 1 Build a U-tree T for $\langle \Sigma, \sigma \rangle$ via η .
- 2 **if** T is trivial **then**
- 3 **return** $LT = \{\}$
- 4 **else**
- 5 Build the minimal E-linearizable upper portion U of T for $\langle \Sigma, \sigma \rangle$ via η .
- 6 **for** leaf l in U which is Eurekaable via ancestor σ' **do**
- 7 Introduce a predicate symbol *eureka* and build $ed \in ED$ as follows:
- 8 $ed : \text{eureka}(\mathbf{X}) \leftarrow I$, where:
 - i) I is a conjunction of IDBs s.t. both the IDBs in l and σ' are instances of I .
 - ii) $\mathbf{X}$ is the minimal subset of variables in I s.t. ed can be folded into l and σ' .
- 9 Remove isomorphic copies from ED
- 10 Select the minimal F-linearizable upper portion U' of U w.r.t. ED .
- 11 **for** ed in ED **do**
- 12 Build the minimal, non-trivial F-linearizable upper portion U_{ed} w.r.t. ED of a U-tree $T' = \langle \Sigma, ed \rangle$ built via η .
- 13 Let LT be the set of all linear TGDs in the leaves of U' and U_{ed} together with the set of all TGDs resulting from the folding of the non-linear TGDs in the leaves of U' and U_{ed} using the TGDs in ED .
- 14 **return** LT

The Proof of Theorem 14 can be derived from Lemmas 11, 12 and 13 along with the steps of Algorithm 1.

Theorem 15 (TLP Correctness). *Let Σ be a PWLW set of TGDs, σ a MNL TGD in Σ and η a linear U-rule for Σ . Then, TLP applied to (Σ, σ, η) terminates. Let LT be the set of linear TGDs returned by TLP, and $pred(\Sigma)$ be the set of predicates defined in Σ , then $M(\Sigma) = M_{pred(\Sigma)}((\Sigma - \{\sigma\}) \cup LT)$.*

Proof Sketch of Theorem 15. Termination: This follows directly from Theorem 14.
Equivalence: It is easy to see that the application of the unfold/fold transformations in TLP complies with the conditions in their definitions. $\square$

By iteratively selecting a MNL TGD in a PWLW set Σ , it is possible to linearize the whole set Σ . We refer to this as *Set Linearization Procedure* (SLP).

Algorithm 2: Set Linearization Procedure (SLP)

Input : a PWLW set of TGDs Σ and a linear U-rule η
Output : a linear PWLW set of TGDs Σ' such that Σ' is CQ-equivalent to Σ

```

1 Apply the Harmful Join Elimination algorithm to  $\Sigma$ , obtaining  $\Sigma^{HJE}$ 
2  $i=0$ 
3  $\Sigma_i^{HJE} = \Sigma^{HJE}$ 
4  $NLT =$  Set of non-linear TGDs
5 while  $NLT$  is not empty do
6   Select a MNL TGD  $\sigma$  from  $NLT$ 
7   Apply TLP to  $(\Sigma_i^{HJE}, \sigma, \eta)$  which returns the set of linear TGDs  $LT_i$ 
8    $\Sigma_{i+1}^{HJE} = (\Sigma_i^{HJE} - \{\sigma\}) \cup LT_i$ 
9    $NLT = NLT - \{\sigma\}$ 
10   $i = i + 1$ .
11  $\Sigma' = \Sigma_i^{HJE}$ 
12 return  $\Sigma'$ 
```

Theorem 16 (SLP Correctness). *Let Σ be a PWLW set of TGDs, η a linear U-rule for Σ , then (i) SLP applied to (Σ, η) terminates. Let LT be the set of linear TGDs returned by SLP applied to (Σ, η) , $pred(\Sigma)$ be the set of predicates defined in Σ , and NLT be the set of all non-linear TGDs in Σ . Then, (ii) $M(\Sigma) = M_{pred(\Sigma)}((\Sigma - NLT) \cup LT)$.*

Proof Sketch of Theorem 16. Termination: since the number of TGDs in NLT is finite and at each iteration one TGD is removed from NLT , SLP terminates.

Correctness: this follows directly from Theorem 14. $\square$

4.2 Linearizability of fuzzy PWLW TGDs

In the previous section, we presented a general linearization procedure for transforming every set of PWLW Datalog[±] programs into a conjunctive-query (CQ)-equivalent linear PWLW Datalog[±] program. In Chapter 3, we emphasized the importance of reasoning in many-valued (MV) settings. We provided extensions of Datalog and Datalog[±] (namely, MV-Datalog and MV-Datalog[±]) specifically designed to handle such scenarios. This section extends our previous work by demonstrating that the linearization procedure can also be effectively applied to optimize reasoning in fuzzy settings. We find that the linearization procedure is suited for optimising the evaluation of K-Truth problems using MV-Datalog and MV-Datalog[±] under the following conditions:

- The program to linearize must be PWLW.
- The chase procedure for the program to linearize must be finite.
- The value of K (i.e. the minimum level of satisfaction of each formula) must be restricted to 1.

The intuition behind this relies on the properties of the operators employed in the Set Linearization Procedure (SLP) outlined in Algorithm 2. These operators, namely fold, unfold, and De-Skolemization, remarkably preserve the dependencies between intensional atoms and extensional atoms in the program to linearize. As these dependencies remain unchanged after linearization, it can be shown that solving a K-Truth problem on the original or linearized program under the above-mentioned conditions retains the same solutions.

This section delves deeper into the details of applying linearization to enhance reasoning in MV settings. We will formally prove the correctness of this approach and showcase its benefits in Section 4.3.

As established earlier, for a given MV-instance (Σ, τ) , the minimal K -fuzzy model can be defined in terms of the linear program $Opt_{\Sigma, \tau}^K$. The same holds for identifying preferred K -fuzzy models for an MV[±]-instance through the linear program $\exists\text{-}Opt_{\Sigma, \tau}^K$. Furthermore, it was previously shown that the variables in both $Opt_{\Sigma, \tau}^K$ and $\exists\text{-}Opt_{\Sigma, \tau}^K$ correspond to the facts appearing in the chase procedure, and these variables do not change during the rewriting algorithm. We start by demonstrating that applying the fold and unfold procedures in both the context of classic and existential Datalog does not alter the solutions of $Opt_{\Sigma, \tau}^K$ nor $\exists\text{-}Opt_{\Sigma, \tau}^K$ for $K = 1$.

Lemma 17. *Let Σ, τ be an MV-instance, $\sigma \in \Sigma$ a rule and idb a non-mutually-recursive predicate symbol appearing in the body of σ . Let Π be the set of rules resulting from $\text{unfold}(\sigma, idb, \Sigma)$. Then $\mathbf{x}$ is an optimal solution of $\text{Opt}_{\Sigma, \tau}^K$ if and only if $\mathbf{x}$ is an optimal solution of $\text{Opt}_{\Sigma', \tau}^K$, where $\Sigma' = \Sigma \cup \Pi \setminus \{\sigma\}$ and $K = 1$.*

Proof of Lemma 17. We can prove this directly. Given an MV-Datalog program Σ , two rules $\sigma, \sigma' \in \Sigma$ and a non-mutually-recursive predicate symbol idb appearing in the body of σ and in the head of only σ' , the equations in $\text{Opt}_{\Sigma, \tau}^K$ for $K = 1$ relative to σ and σ' are:

$$\text{Eval}(\sigma) := \sum_{b=1}^{\ell} (1 - x_b) + x_{head} \geq 1 \quad (4.1)$$

$$\text{Eval}(\sigma') := \sum_{b'=1}^{\ell'} (1 - x_{b'}) + x_{head'} \geq 1 \quad (4.2)$$

where ℓ, ℓ' are the indexes of the truth values variables associated with body atoms of σ and σ' and $head, head'$ are the indexes of the truth values variables associated with the head atoms of σ and σ' . If idb is associated to x_b , with $b = 1$ in 4.1 and to $x_{head'}$ in 4.2, then the following holds:

$$x_{head'} \geq 1 - \sum_{b'=1}^{\ell'} (1 - x_{b'}) \quad (4.3)$$

For the moment, let's assume that σ' is the only rule in which $x_{head'}$ appears. In this case, considering that the optimization function of $\text{Opt}_{\Sigma, \tau}^K$ minimizes all the variables, we can replace 4.3 with:

$$x_{head'} = 1 - \sum_{b'=1}^{\ell'} (1 - x_{b'}) \quad (4.4)$$

and substitute the value of $x_{head'}$ from 4.4 inside 4.1:

$$\text{Eval}(\sigma) = \sum_{b=2}^{\ell} (1 - x_b) + x_{head} + 1 - \left(1 - \sum_{b'=1}^{\ell'} (1 - x_{b'})\right) \geq 1 \quad (4.5)$$

$$\sum_{b=2}^{\ell} (1 - x_b) + x_{head} + \sum_{b'=1}^{\ell'} (1 - x_{b'}) \geq 1 \quad (4.6)$$

In case $x_{head'}$ appears also in the head of another rule $\sigma^* \in \Sigma$, with:

$$\text{Eval}(\sigma^*) := x_{head'} \geq 1 - \sum_{b^*=1}^{\ell'} (1 - x_{b^*}) > 1 - \sum_{b'=1}^{\ell'} (1 - x_{b'}) \quad (4.7)$$

the solutions of $x_{head'}$ would change and lead to different solutions of x_{head} . Though, since $\sigma^* \in \Sigma$, also $\text{Eval}(\sigma^*)$ is part of the inequalities of $\text{Opt}_{\Sigma, \tau}^K$. Therefore, the new minimum value of $x_{head'}$ is:

$$x_{head'} = 1 - \sum_{b^*=1}^{\ell'} (1 - x_{b^*}) \quad (4.8)$$

which can be replaced inside 4.1 to obtain:

$$\sum_{b=2}^{\ell} (1 - x_b) + x_{head} + \sum_{b^*=1}^{\ell'} (1 - x_{b^*}) \geq 1 \quad (4.9)$$

Note that $x_{head'}$ might also appear in the body of some rule ρ . However, the unfold operator does not select ρ as it selects only the rules whose head matches with $x_{head'}$. Therefore, the equations associated with such rules remain the same. When the unfold operator is applied to σ , obtaining σ'' , the following holds:

$$\text{Eval}(\sigma'') := \sum_{b=2}^{\ell} (1 - x_b) + x_{head} + \sum_{b'=1}^{\ell'} (1 - x_{b'}) \geq 1 \quad (4.10)$$

Where, in the case where $\sigma^* \notin \Sigma$, $\text{Eval}(\sigma) = \text{Eval}(\sigma'')$. In case where there exists a $\sigma^* \in \Sigma$ for which 4.7 holds, $\text{Eval}(\sigma) \neq \text{Eval}(\sigma'')$. Though, the unfold procedure would return also:

$$\text{Eval}(\sigma''^*) := \sum_{b=2}^{\ell} (1 - x_b) + x_{head} + \sum_{b^*=1}^{\ell^*} (1 - x_{b^*}) \geq 1 \quad (4.11)$$

such that $\text{Eval}(\sigma) = \text{Eval}(\sigma''^*)$. □

We now extend the result of Lemma 17 to Datalog[±]:

Lemma 18. *Let Σ, τ be an $MV^\pm$ -instance, $\sigma \in \Sigma$ a rule and idb a non-mutually-recursive predicate symbol appearing in the body of σ . Let Π be the set of rules returned by $\text{unfold}^*(\sigma, idb, \Sigma)$. Then $\mathbf{x}$ is an optimal solution of $\exists\text{-Opt}_{\Sigma, \tau}^K$ if and only if $\mathbf{x}$ is an optimal solution of $\exists\text{-Opt}_{\Sigma', \tau}^K$, where $\Sigma' = \Sigma \cup \Pi \setminus \{\sigma\}$ and $K = 1$.*

Proof of Lemma 18. This builds upon the Proof of Lemma 17. Since the unfold procedure in the existential case does not add new atoms to the body (modulo the Skolem functions that will be simplified), it remains to prove that applying the De-Skolemization procedure does not change the results of $\exists\text{-Opt}_{\Sigma,\tau}^K$. When unfolding on an atom in $\sigma \in \Sigma$ with predicate name `idb` using σ' , only the following three possible cases exist:

- one of the existentially quantified variables appearing in the atom with predicate name `idb` in the body of σ is involved in a harmless join. In this case, the De-Skolemization simply eliminates the resulting rule, as this would never find any acceptable grounding in either Σ or Σ' .
- the existentially quantified variables appearing in the atom with predicate name `idb` in the body of σ are not involved in harmless joins nor present in the head of σ . In this case, the existentially quantified variables are replaced by a fresh value, not changing the number of groundings of `idb`, nor the truth values that are associated with this one. Therefore, the De-Skolemization does not change the formulation of the Eval constraints after the unfolding.
- the existentially quantified variables appearing in the atom with predicate name `idb` in the body of σ are not involved in harmless joins and are propagated to the head of σ . In this case, in order to preserve the original number of groundings of `idb`, new rules σ'' and σ''' are introduced to obtain a level of indirection. The forms of these two are shown in the De-Skolemization procedure. In the following, we prove directly that substituting σ with σ'' and σ''' leads to the same solutions of $\exists\text{-Opt}_{\Sigma,\tau}^K$ for $K = 1$. We here report the inequality 4.6 corresponding to σ reported in the Proof of Lemma 17:

$$\text{Eval}(\sigma) := \sum_{b=2}^{\ell} (1 - x_b) + x_{\text{head}} + \sum_{b'=1}^{\ell'} (1 - x_{b'}) \geq 1 \quad (4.12)$$

where ℓ, ℓ' are the indexes of the truth values variables associated with body atoms of, respectively, σ and σ' , $\text{head}, \text{head}'$ are the indexes of the truth values variables associated with the head atoms of, respectively, σ and σ' . We now compute $\text{Eval}(\sigma'')$ and $\text{Eval}(\sigma''')$ resulting from the unfold operation:

$$\text{Eval}(\sigma'') := \sum_{b=2}^{\ell} (1 - x_b) + x_{\text{new}} + \sum_{b'=1}^{\ell'} (1 - x_{b'}) \geq 1 \quad (4.13)$$

$$\text{Eval}(\sigma''') := 1 - x_{\text{new}} + x_{\text{head}} \geq 1 \quad (4.14)$$

where x_{new} is the variable associated with the truth value of the newly introduced predicate used in place of x_{head} to obtain a level of indirection. The following holds from $\text{Eval}(\sigma'')$:

$$x_{new} \geq 1 - \sum_{b=2}^{\ell} (1 - x_b) - \sum_{b'=1}^{\ell'} (1 - x_{b'}) \quad (4.15)$$

Since x_{new} is a variable associated with a fresh predicate, the optimization function of $\exists\text{-Opt}_{\Sigma, \tau}^K$ will minimize it, we know that:

$$x_{new} = 1 - \sum_{b=2}^{\ell} (1 - x_b) - \sum_{b'=1}^{\ell'} (1 - x_{b'}) \quad (4.16)$$

which we can substitute in $\text{Eval}(\sigma''')$ to obtain:

$$1 - \left(1 - \sum_{b=2}^{\ell} (1 - x_b) - \sum_{b'=1}^{\ell'} (1 - x_{b'})\right) + x_{head} \geq 1 \quad (4.17)$$

$$\sum_{b=2}^{\ell} (1 - x_b) + \sum_{b'=1}^{\ell'} (1 - x_{b'}) + x_{head} \geq 1 \quad (4.18)$$

Note that the De-Skolemization preserves the number of labelled nulls that would be instantiated before and after the unfold procedure, therefore the number and values of the Eval inequalities do not change in any of the aforementioned cases. $\square$

We also show that an analogous outcome holds for the fold operation, dual to the unfold one.

Lemma 19. *Let Σ, τ be an MV-instance, $\sigma \in \Sigma$ a rule, Φ a set of non-mutually-recursive atoms in $\text{body}(\sigma)$ and $\sigma' \in \Sigma$ the only rule such that $\text{body}(\sigma')\theta = \Phi$ and there is no other rule ρ in Σ such that there exists θ' such that $\text{head}(\rho)\theta' = \text{head}(\sigma')$. Let Π be the set of rules resulting from $\text{fold}(\sigma, \sigma', \Phi, \Sigma)$. Then $\mathbf{x}$ is an optimal solution of $\text{Opt}_{\Sigma, \tau}^K$ if and only if $\mathbf{x}$ is an optimal solution of $\text{Opt}_{\Sigma', \tau}^K$, where $\Sigma' = \Sigma \cup \Pi \setminus \{\sigma\}$ and $K = 1$.*

Proof of Lemma 19. This can be proven directly similarly to Lemma 17 and 18. Given an MV-Datalog program Σ , two rules $\sigma, \sigma' \in \Sigma$ where $\Phi \in \text{body}(\sigma)$, $\text{body}(\sigma')\theta =$

Φ and there is no other rule in Σ such that there exists θ' such that $head(\rho)\theta' = head(\sigma')$. The equations in $Opt_{\Sigma, \tau}^K$ for $K = 1$ relative to σ and σ' are:

$$Eval(\sigma) := \sum_{b=1}^{\ell} (1 - x_b) + \sum_{b'=l+1}^{\ell'} (1 - x_{b'}) + x_{head} \geq 1 \quad (4.19)$$

$$Eval(\sigma') := \sum_{b=1}^{\ell} (1 - x_b) + x_{head'} \geq 1 \quad (4.20)$$

where ℓ, ℓ' are the indexes of the truth values variables associated with body atoms of, respectively, σ and σ' , and $head, head'$ are the indexes of the truth values variables associated with the head atoms of, respectively, σ and σ' . Note that the indexes in 4.19 are reused in 4.20 as they refer to the group of atoms in Φ which can be unified through θ . From 4.20 it follows:

$$x_{head'} \geq 1 - \sum_{b=1}^{\ell} (1 - x_b) \quad (4.21)$$

Since $x_{head'}$ is unique and the optimization problem will minimise it:

$$x_{head'} = 1 - \sum_{b=1}^{\ell} (1 - x_b) \quad (4.22)$$

$$\sum_{b=1}^{\ell} (1 - x_b) = 1 - x_{head'} \quad (4.23)$$

Which we can substitute into 4.19 to obtain:

$$1 - x_{head'} + \sum_{b'=l+1}^{\ell'} (1 - x_{b'}) + x_{head} \geq 1 \quad (4.24)$$

$$\sum_{b'=l+1}^{\ell'} (1 - x_{b'}) + x_{head} - x_{head'} \geq 0 \quad (4.25)$$

The Eval associated to the result of $fold(\sigma, \sigma', \Phi, \Sigma)$, where Φ is the set of atoms to which variables x_b with $l \geq b \geq 1$ in 4.19 and 4.20 is the following:

$$Eval(\sigma'') := 1 - x_{head'} + \sum_{b'=l+1}^{\ell'} (1 - x_{b'}) + x_{head} \geq 1 \quad (4.26)$$

$$\sum_{b'=l+1}^{\ell'} (1 - x_{b'}) + x_{head} - x_{head'} \geq 0 \quad (4.27)$$

We can observe that 4.25 is equivalent to 4.27. Therefore, since the application of the fold procedure does not change the resulting Eval equations, we have that an optimal solution to $Opt_{\Sigma, \tau}^K$ coincides with an optimal solution to $Opt_{\Sigma', \tau}^K$. $\square$

This result can also be extended to the existential case.

Lemma 20. *Let Σ, τ be an $MV^\pm$ -instance, $\sigma \in \Sigma$ a rule, Φ a set of non-mutually-recursive atoms in $body(\sigma)$ and $\sigma' \in \Sigma$ the only rule such that $body(\sigma')\theta = \Phi$ and there is no other rule ρ in Σ such that there exists θ' such that $head(\rho)\theta' = head(\sigma')$. Let Π be the set of rules resulting from $fold^*(\sigma, \sigma', \Phi, \Sigma)$. Then x is an optimal solution of $Opt_{\Sigma, \tau}^K$ if and only if x is an optimal solution of $Opt_{\Sigma', \tau}^K$, where $\Sigma' = \Sigma \cup \Pi \setminus \{\sigma\}$ and $K = 1$.*

The proof for Lemma 20 is exactly the same of Lemma 19.

Building on the results of Lemma 18 and Lemma 20 we can formulate the main theorem of this section:

Theorem 21. *Let Σ, τ be a PWLW MV-Datalog program and database where $OLim(\Sigma^{crisp}, D_\tau)$ is finite. Let Σ' be the result of $SLP(\Sigma, \eta)$ for some linear U-rule η . If $\exists Opt_{\Sigma, \tau}^K$ is feasible, then v_x is an optimal solution to $\exists Opt_{\Sigma, \tau}^K$ iff v_x is an optimal solution to $\exists Opt_{\Sigma', \tau}^K$.*

The proof of Theorem 21 follows from those of Lemma 18 and Lemma 20 as the in the SLP procedure atomic steps are composed of a combination of the *unfold** and *fold** procedures which do not tamper the solutions of $\exists Opt_{\Sigma, \tau}^K$.

Theorem 21 guarantees that the linearized program produced by the procedure retains the same K-truth evaluation behaviour as the original program. This allows us to use the linearization procedure to optimise the evaluation of K-truth problems without requiring any alterations to the procedure itself.

4.3 Implementation and Experiments

In this section, we describe the implementation of the linearization procedure in a dedicated processing module in the Vadalog system [25], and we show how this system exploits such a normal form to provide a fully streaming and space-efficient evaluation of PWLW TGDs.

4.3.1 Vadalog System

We now introduce Vadalog, the system utilized for conducting the experiments presented below. The Vadalog system is a state-of-the-art reasoner whose core language is based on Warded Datalog[±]. It natively supports ontological query answering, probabilistic reasoning, and numeric computations (i.e., standard and monotonic aggregations). Let us briefly describe its architecture.

System Architecture. Given a set of TGDs Σ and a CQ $Q : q(\mathbf{x}) \leftarrow \phi(\mathbf{x}, \mathbf{y})$, an active pipeline is compiled out of them, where atoms correspond to filters, connected by pipes that denote the input-output transformations applied by the TGDs. EDBs are mapped onto the pipeline sources and $q(\mathbf{x})$ corresponds to the output filter. Data flow from sources filters, i.e., the EDBs, to the target, i.e., the query, undergoing the transformations (e.g., selections, projections, joins, value inventions) performed by the rules. Finally, the facts returned by the output filter are those composing the CQ answer. This process is implemented by three dedicated architectural components.

1. The *logic optimizer* performs rewritings and syntactic condition checks. This module determines whether a set of TGDs Σ is PWLW by performing syntactic condition checks. Then, it rewrites Σ into an equivalent set Σ' of Afrati-linear warded TGDs by executing the Algorithm 2.
2. The *logic compiler* transforms Σ' into a reasoning access plan: it produces a pipeline in the form of a predicate graph where each node (filter) represents an atom and there is an edge (pipe) from nodes m to n if there is a TGD with m in the body and n in the head. Extensional atoms are mapped into the pipeline sources and $q(\mathbf{x})$ corresponds to the output filter.
3. The *query compiler* converts the logic pipeline into a *reasoning query plan*, where the nodes are translated into active *data scans*, connected by buffers. In our architecture, we have three different types of scan: a linear scan for linear TGDs, a join scan for join TGDs, and an output scan for the query.

Execution Model. It relies on a *pull-based* (query-driven) approach, where each scan reads facts from the parents. Such an implementation is a generalization of the *volcano iterator model* [79] and allows to generate only the facts required to answer a query without materializing all the IDBs in Σ . The pull action starts from the output scan (i.e., corresponding to the CQ atom $q(x)$), which asks for output facts and propagates the request down to its predecessors, triggering the invocations along the pipeline searching for available facts. Scans interact with each other using primitives `open()`, `next()`, `get()`, `close()`, which open the parent stream and ask for the presence of new facts via the `next()` primitive, obtain it, and close the communication. Each scan is equipped with an indexed cache that stores the newly generated facts and can provide them in the case of multiple `next()` calls by different consumers. In order to block the generation of an already produced fact, all the scans for the same predicate are equipped with a Bloom filter to perform memory-preserving duplication checks.

Join scans are connected to two sources and are implemented with a *nested loop*: when the first `next()` call is sent to a join scan, its right-hand side source scan is triggered to fixpoint until all the facts available are generated (and an index on the join key is built). Then, each `next()` call on the join scan asks for facts to the left-hand side until a match is found and a new fact is returned.

Exploiting Linearity. When the program is non-linear, every join scan reading only from IDBs must wait for the materialization of one or more of the predicates associated with one of its (non-recursive) sources. This introduces a blocking synchronization step for the execution pipeline and a potential memory overload. On the other hand, when the program is linear, Vadalog optimizes the execution by implementing a fully streaming and memory-bound evaluation. This is possible for the following reasons: (i) linear scans perform fact-independent operations (e.g., filters, selections, constants creation) and do not require any synchronization; (ii) join scans only compare streams of tuples (IDBs) with static and not-evolving (in terms of newly generated facts) sources (EDBs). Before the program evaluation starts, each EDB that is a source for a join scan is loaded into the main memory and indexed on the corresponding join keys. The buffer of each scan uses only a constant portion of memory: whenever a new fact is stored in a buffer, it is consumed by all the parent scans and then removed from the memory. This implies that when the program is linear, the system uses only a constant portion of memory for the program evaluation of each intermediate scan.

Currently, no parallel join is implemented in Vadalog. Adding this would increase

the positive impact of the linearization on the execution times. Moreover, linearized versions of programs present rules with more predicates in the bodies, i.e. more joins. This allows for a more extensive use of multi-way joins rather than the simple use of the more inefficient sequence of binary joins. Finally, having a single multi-way join rather than a sequence of binary join enforced by the program structure, would allow the optimization of join operators by pushing the smaller relations before in the evaluation of the join.

4.3.2 Experimental Evaluation

This section investigates the performance impact of the linearization procedure on CQ-answering using the Vadalog system. We employ a series of synthetic scenarios with a synthetic fuzzy dataset to evaluate the procedure's effectiveness. While our current implementation emphasizes the linearization algorithm itself, incorporating additional components, such as a parallel join operator, could further enhance the system and comprehensively demonstrate the benefits of linearizing PWLW programs for efficient CQ-answering.

Test Setup. All external input and output sources have been stored in CSV files, all scenarios involve sets of PWLW TGDs. We ran each experiment 5 times, averaging the elapsed times. We used a remote instance of the Vadalog system, running on a server with Ubuntu v20.04 in a Linux machine with eight physical 2.1 GHz Xeon v4 cores, 150 GB of RAM, and a 1 TB SSD.

We isolated the logic optimizer module as an independent component for the linearization experiments. The evaluation experiments include the following steps: (1) load the TGDs from a textual file and the source instance from CSV files; (2) run the comprehensive pipeline which is compiled from both Σ and the rule σ_Q , representing the CQ Q . In the experiments involving execution times, also the pre-loading into main memory and indexing process of EDB predicates is considered.

Synthetic Scenarios: iWarded. To study the impact of the properties of warded TGDs on the performance of the linearization algorithm implemented in Vadalog, we used iWarded [15], a generator of benchmarks for warded TGDs. This controls the fragment internals, such as the number of linear, non-linear join and warded join TGDs, the number of mutually recursive predicates and the presence of existential quantification.

Varying Fragment Internals. We measured the impact of linearization on six different synthetic scenarios, each composed of 100 rules, using a combination of 50 input CSV files containing 1000 data points each, augmented with a random truth value between 0 and 1. The synthetic programs we adopted are the same as described in the paper that presented Vatalog [30]. Here, we briefly describe the synthetic programs.

SynthA has a prevalence of linear swap rules and 20% of the total rules have existential quantification; 30% of linear and non-linear rules are recursive, and the joins are equally distributed between harmless-harmful and harmless-harmless. *SynthB* is specular to *SynthA*, but has a prevalence of non-linear rules; all the other proportions are respected accordingly. *SynthC* (as well as all the following settings) has a 30%-70% balance between linear and non-linear rules with the presence of harmful-harmful joins. *SynthD* stresses the presence of such joins even more. *SynthE* studies the impact of a strong presence of recursion of the non-linear rules. *SynthF* deals with a high recursion incidence on linear ones. *SynthG* has basically the same characteristics as *SynthC* and *SynthD*, but with a prevalence of harmless rules, therefore its characteristics resemble those of a pure Datalog program. *SynthH* emphasizes the joins on wards. Further details on these scenarios can be found in Table 4.1

Program	L rules	$\bowtie$ rules	L rec	$\bowtie$ rec	$\exists$ rules	hrml $\bowtie$ hrmf	hrml $\bowtie$ hrml with ward	hrml $\bowtie$ hrml w/o ward	hrmf $\bowtie$ hrmf
<i>synthA</i>	90	10	27	3	20	5	4	1	0
<i>synthB</i>	10	90	3	27	20	45	40	5	0
<i>synthC</i>	30	70	9	20	40	25	20	5	20
<i>synthD</i>	30	70	9	20	22	10	9	1	50
<i>synthE</i>	30	70	15	40	20	35	29	1	5
<i>synthF</i>	30	70	25	20	50	35	29	1	5
<i>synthG</i>	30	70	9	21	30	0	10	60	0
<i>synthH</i>	30	70	9	21	30	0	60	10	0

Table 4.1

Details of the scenarios generated with iWarded (Table taken from [30])

For each scenario, Figure 4.3 reports the results of the test executions. It is possible to notice that the linearization reduces the execution time in all scenarios but *SynthA* and *SynthC*. Notably, we couldn't reproduce the execution of scenarios *SynthF* and *SynthG*. Figure 4.4 compares the execution of *SynthA* and *SynthC* with *SynthA'* and *SynthC'* which are obtained by removing all the swap rules. We notice a drastic change in the performances of the linearized *SynthC'*. On the other hand, *SynthA'* continues to show a performance drop when linearized. We rooted this to a specific optimization hardcoded in Vatalog which recognizes

specific patterns of programs and optimizes them. In fact, it is possible to notice that the linearized *SynthA'* reports the same evaluation time that was reported in [30].

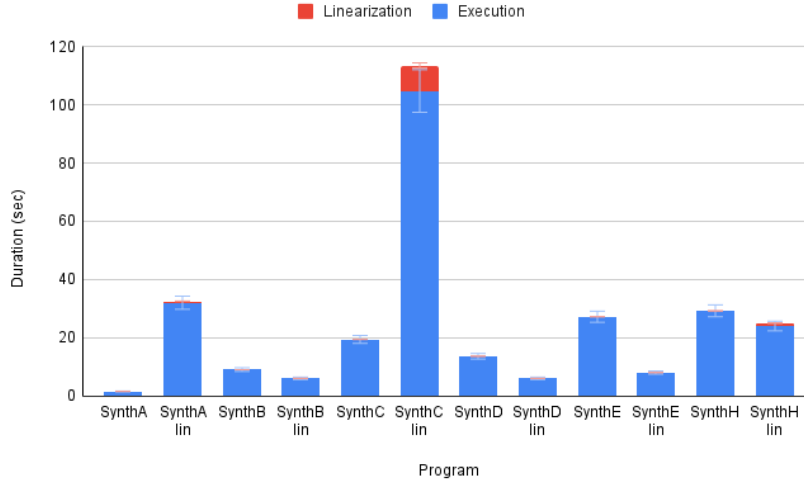


Figure 4.3: Execution times of the original synthetic programs [30] with and without linearization.

Varying Number of Swaps. In this scenario, we test the impact of variable swaps. A variable swap is a displacement of a variable position obtained through a rule of the form $idb(x, y) \rightarrow idb(y, x)$. This uncommon pattern negatively impacts linearization when unfolding over a predicate involved in swaps, as it requires introducing a number of linear rules that are exponential in the number of variables in the swaps. This is due to the necessity of introducing a number of linear rules that are exponential in the number of variables involved in swaps in order to fully represent the program's original structure. As shown in Figure 4.5, increasing the number of swaps significantly impacts the linearization time. Moreover, as shown in Figure 4.4, this negatively impacts also the execution times of the linearized programs. Nevertheless, this case seems unrealistic in real-world scenarios.

4.3.3 Discussion and $PWLW^-$

Our experiments demonstrate that the linearization improves the performance of CQAns for most $PWLW$ programs (see Figure 4.4). However, we observed that programs containing variables' swaps operations present performance degradation once linearized. To address this, we introduce the fragment $PWLW^-$.

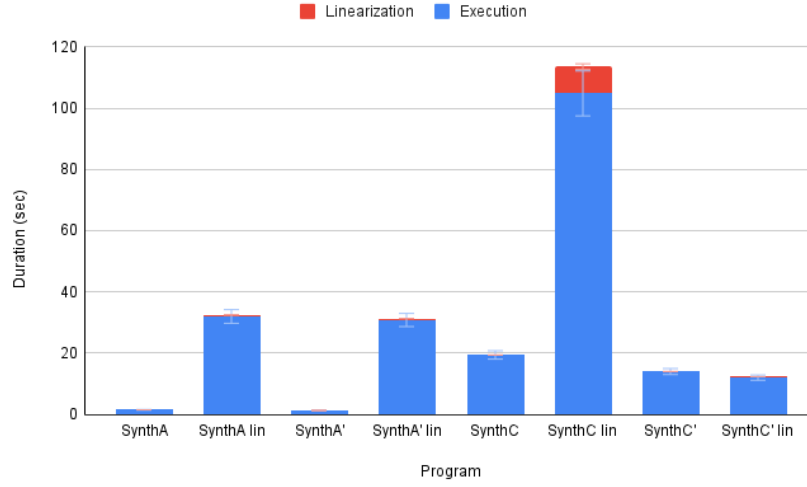


Figure 4.4: Execution times of *SynthA'* and *SynthC'* with and without linearization compared to the original *SynthA* and *SynthC*.

Definition 39. A program Σ is a PWLW^- program if the following holds:

- Σ is a PWLW program.
- Σ does not contain any rules with variable swap operations.

By applying the linearization procedure only to a specific subset of rules belonging to the PWLW^- fragment, we can ensure consistent improvement in efficiency for the CQAns task across all PWLW programs. This selective approach avoids the performance degradation observed for programs with variable swaps.

It is important to note that the performance improvements achieved by our linearization technique were obtained without the use of ad-hoc parallel operators in Vadalogue. This is significant because implementing such operators would likely result in greater performance benefits. Therefore, our results demonstrate our linearization approach's effectiveness, even without specialized parallelization techniques. However, future work could explore the integration of ad-hoc parallel operators with our linearization technique to further improve the CQAns task's efficiency.

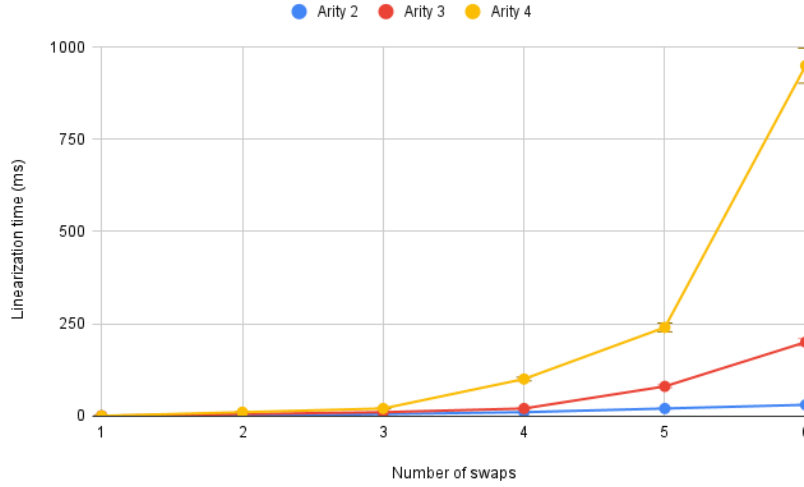


Figure 4.5: Impact of the number of swaps on linearization times.

4.4 Summary

This chapter presents a novel approach to optimizing logical reasoning for programs in the PWLW Datalog[±] fragment. We prove that PWLW Datalog[±] programs can be rewritten into a normal form, which we refer to as Afrati-linear, to distinguish it from the usual connotation of linearity. The key feature of Afrati-linear programs is restricting the rules to contain at most one intensional atom in their body.

To linearise PWLW Datalog[±], we extend the well-established fold and unfold operators for Datalog to existential settings and introduce a new procedure called De-Skolemization. These techniques can be used to reduce the number of intensional atoms in the body of the rules while preserving the program's semantics. By iteratively applying such procedure on specific rules, referred to as minimally non-linear, it is possible to linearise PWLW programs. We present procedures for linearising individual TGDs and entire programs, called the TGD Linearization Procedure (TLP) and Set Linearization Procedure (SLP).

Linearization offers several benefits. It reveals the chain of joins hidden by intensional atoms, facilitating pattern recognition and case-specific optimizations. Additionally, the Afrati-linear form allows for efficient evaluation by materialising all the extensional atoms in a rule's body beforehand and evaluating them against the single intensional atom. Furthermore, linearization reduces inter-rule depen-

dencies, enabling the clustering of the rules into layers that can be evaluated independently.

We extend the linearization procedure to many-valued settings with MV-Datalog and MV-Datalog[±] under certain conditions. We prove that both the classic and our extensions of the fold, unfold, and De-Skolemization operators preserve solutions for linear programs used in K-truth problem computations. This allows the linearization procedure to be reused in many-valued settings without requiring ad-hoc considerations.

Finally, we describe the implementation of the linearization procedure in the Vatalog reasoner. This implementation leverages the Afrati-linear form to enable a fully streaming and memory-bound reasoning architecture. Experimental evaluation demonstrates performance improvements for CQ answering on various fuzzy data scenarios. We introduce a new fragment, PWLW⁻, which excludes rules with variable swaps, further ensuring consistent performance gains for both crisp and fuzzy settings.

CHAPTER 5

Automatization Of Datalog Rules Discovery and Application

Relevant applications of Datalog to real-world problems include the completion (also referred to as link prediction/record completion, see [116]) and correction (also referred to as smell/outlier detection, see [167]) of data in Knowledge Graphs.

Writing logical rules to address such problems is a never-ending and expensive task requiring interactions between Datalog experts and domain experts. These will attempt to model the domain as much as possible, though it is almost impossible to write logical rules covering all the possible corner cases.

Example 18. Let us consider, for example, the problem of determining whether or not it is safe to grant a loan to a person A . A domain expert may argue that if a person A lives with a person B and B is a bad creditor, then A might be a bad creditor as well. This can be expressed through the Datalog rule σ :

$$\text{lives}(P_1, \text{Addr}), \text{lives}(P_2, \text{Addr}), \text{reliable}(P_2, \text{false}) \rightarrow \text{reliable}(P_1, \text{false}).$$

After further reasoning, the domain expert might suggest that the relationship between A and B should also be taken into account as, if they are simply acquaintances sharing a flat, this might unfairly discriminate A . Then, σ can be refined

into the following rules:

$$\begin{aligned} & \text{lives}(P_1, \text{Addr}), \text{lives}(P_2, \text{Addr}), \text{reliable}(P_2, \text{false}), \text{friends}(P_1, P_2) \\ & \quad \rightarrow \text{reliable}(P_1, \text{false}). \\ & \text{lives}(P_1, \text{Addr}), \text{lives}(P_2, \text{Addr}), \text{reliable}(P_2, \text{false}), \text{relatives}(P_1, P_2) \\ & \quad \rightarrow \text{reliable}(P_1, \text{false}). \end{aligned}$$

Yet other corner cases might consider the eventuality that A or B no longer live at the indicated address or that A has a strong credit history that should overrule the unreliability given by the co-habitation with B .

It's easy to capture the challenge of individuating all the relevant conditions that might be considered in Example 18. The area of Inductive Logic Programming (described in Section 2.6) addresses this problem by learning logical rules from Big Data. ILP techniques search for statistical evidence in the data to discover logical rules. In this chapter, we focus on the association rules mining family of ILP techniques. These techniques are used to discover association rules, i.e. logical rules expressing the correlation between the attributes of a dataset.

Contributions In this chapter, we introduce a patented method [151] for mining and applying association rules to detect data outliers and complete database records in the context of Big Data and many-valued settings. The presented method is named $ARM(D, I)$, where D is a target database and I is a set of user-defined inputs. The underlying principles of $ARM(D, I)$ can be abstractly summarized as follows:

1. The data in the target database D are prepared for effective and efficient mining. This includes, where appropriate, the removal of all the columns in D which would have brought no benefit to the mining of association rules. Moreover, where D contains non-crisp truth values, minimum thresholds on attribute-level and record-level are applied. In the eventuality that the columns in D contain truth degrees referring to either the records in D or other columns in D and these truth degrees are not utilised in later processes, they are also removed.
2. The data in D are utilised to mine frequent itemsets from each of which several association rules are generated. Both the frequent itemsets and the association rules are subject to minimum values thresholding of, respectively,

support and confidence to ensure a high quality of the generated rules. Moreover, the volume of the association rules generated is controlled by means of minimum and maximum target values. Optionally, the truth values of the columns in D are exploited during the mining process to obtain more accurate resulting rules.

3. The set of computed association rules AR is subject to statistical relevance tests to further assess its quality. This is carried out by computing a combination of metrics based on AR and D . These metrics are then used to filter out low-quality rules from AR .
4. Optionally, where a set of domain experts is available, the association rules AR are evaluated by a crowd of experts. Several methods to determine the number of rules and the selection of the expert reviewing such rules are employed. A metric to aggregate and decide the success of the human evaluation is computed.
5. An optional decision-tree-like structure is created from the association rules AR and is examined by the crowd of experts as an ulterior quality check.
6. The association rules AR are employed in several tasks to improve the quality of the original database D and/or of a database D' , unrelated to $ARM(D, I)$, with a schema equivalent to the schema of D .
7. Several mechanisms are utilised to automatically halt the computation and/or restart it with an automatically adjusted set of input parameters.

We also apply the methodologies of $ARM(D, I)$ to an industrial use case using a Knowledge Graph containing companies' data. We show the effectiveness of the methodology and the results of the evaluation made by experts.

5.1 System Description

In this section, we introduce an association rules-based data outlier detection and record completion system and method in the context of many-valued settings. These are also illustrated by using a running example to help the readability. Figure 5.1 shows an overview of the system, including all the subprocesses. Here, each element is labeled with a number that will be used as a reference throughout this section. For a better visualization, we provide Figures A.1,A.2,A.3,A.4,A.5,A.6 depicting subportions of Figure 5.1 in the appendix to this document. The computation starts from the element **103** and stops when element **104** is reached. In addition, examples of variations and/or extensions are discussed with each process.

Association rules are generated from a database with possibly incomplete, erroneous or non-crisp (fuzzy) data. The generated rules are applied to the records in the database to detect outliers (and thus possibly erroneous data items in the database) and/or to complete records with missing and/or partial information.

We present a method $ARM(D, I)$ for mining association rules AR from an input database D , and a set I of input parameters, some of which express constraints on the quantity and quality of the rules to be mined. The input database D is an exact copy of another database D_0 , where the original data are stored and protected from modifications and failures that might occur during $ARM(D, I)$.

The set of mined association rules AR expresses data dependencies over the database D . D may contain fuzzy data, where rational truth values from the interval $[0,1]$ augmented by a null value such as, for example, "NULL" are associated with data values. Here, 0 stands for "false", 1 stands for "true", and a null value means the fuzzy truth value is unknown. A data value not associated with a fuzzy value has, by default, a truth value equal to 1. Thus, a traditional database corresponds to a fuzzy database where each data value has the crisp truth value 1.

The input required by the method ARM is:

- an input database D_0 comprising a single relational table over a schema A_0 of attributes, where A_0 comprises a non-empty set of regular attributes A_0^r , and for each non-crisp attribute $a \in A_0^r$, A_0 also includes a truth-value attribute coupled to a , referred to as t_a , ranging over a closed interval $[0,1]$ of rational numbers, such that each data value stored in the a -component of each tuple

t of D_0 is associated with a truth value stored in the t_a -component of t , and where data values in columns corresponding to attributes not coupled with a truth-value attribute are, by default, associated with a truth value 1, and

- a set I of input parameters composed of several other input parameters. This will be thoroughly detailed further below.

The procedure must also have access to a database E containing, for each expert e involved in the reviewing process of the current or past iterations of the procedure, information about their revisions history. The schema of E is the following: $(expert_id, attribute, \#checks)$. Here, the values of the $expert_id$ attribute are unique identifiers associated with each expert e , the values of the attribute $attribute$ are the attributes' names of D and, in each tuple of E having $expert_id = e$ and $attribute = a$, the value of $\#checks$ is the number of checks involving a rule from AR containing attribute a that expert e has made so far.

The main features of the method include:

1. Making a copy D of the input database D_0 , where the schema A of D is initially identical to A_0 . The method subsequently operates on the database D guided by the input parameters.
2. Methodologies of data preparation to eliminate attributes from A that are deemed irrelevant to the mining, together with the corresponding columns from D ; to remove from D records whose aggregated fuzzy truth value is below a constant specified in the input I ; and to perform, guided by input parameters, discretizations of domains and data values of certain attributes.
3. Algorithms of association rules mining applied to D used to generate the set of association rules AR . For each $ar \in AR$ the method computes and associates a support $S(ar)$ and a confidence $C(ar)$.
4. Methodologies of rule cleaning to eliminate statistically insignificant rules from the set of mined rules.
5. Methodologies of rule evaluation to which generated association rules are sent for assessment to process at least a subset of evaluations.
6. Methodologies for automated rule application, including (a) outlier detection by identifying records and attribute values that violate one or more association rules and that correct the records, and (b) truth reinforcement,

comprising raising the truth value, in case the truth value is less than 1, of data values in D that concord with values predicted by applicable association rules, and (c) data completion in which values predicted by the association rules replace null values.

When the steps of human validation are enabled, one or more domain experts perform a review as part of the process, possibly combined with automated rule validation. In some examples, once all the human experts involved have terminated their evaluation of the rule set AR , a trigger is activated to either restart the computation with an automatic process to increase the minimum values in I to produce more restrictive rules or to confirm the satisfactory quality of the produced rules AR . Every time that a diamond element is utilised in the diagrams represented in the figures (such as **123**, **144**, **151** and **152** appearing in Figure 5.1), the computation may proceed following one of two different paths.

5.1.1 Input Parameters (102)

In the following, we describe the input I (**102**) to the system. This has been designed to allow for flexibility and easy extensibility of the functionalities.

1. Set FD of *non-trivial functional dependencies (FDs)* of A of the form $L \rightarrow R$ where $L, R \subseteq A$ are possibly compound attributes of A . The set FD is supposed to be a cover that logically entails all valid functional dependencies holding over A in a given context. If not provided, a default value of $FD = \emptyset$ is used.
2. Set $U \subseteq A$ of known *uninteresting attributes* from A . An attribute a is considered uninteresting if it is undesired that it appears in the body of an association rule. If not provided, a default value of $U = \emptyset$ is used.
3. Set $Int \subseteq (A \setminus U)$ of known *interesting attributes* to be mined over A , where an attribute a is interesting if the user is interested in finding rules $a_1, \dots, a_r \rightarrow a$. If not provided, a default value of $Int = \emptyset$ is used.
4. Set $B = \{b_1[\sigma_1], \dots, b_d[\sigma_d]\}$ of attributes in A that must be *discretized*. For any attribute b_i where $1 \leq i \leq d$, σ_i is a method specifier, that is, $\sigma_i \in \mathbb{N} \cup \{true\} \cup MF$. σ_i specifies how the values of attribute b_i of relation schema A ought to be discretized, where $MF = \{mf_1, \dots, mf_n\}$. $mf_i \in MF$ is a data structure of the form $\{mf, q\}$ where mf is a membership function that maps the value of the attribute $b_i \in B$ to a class and $q \in \mathbb{N}$. In particular, if

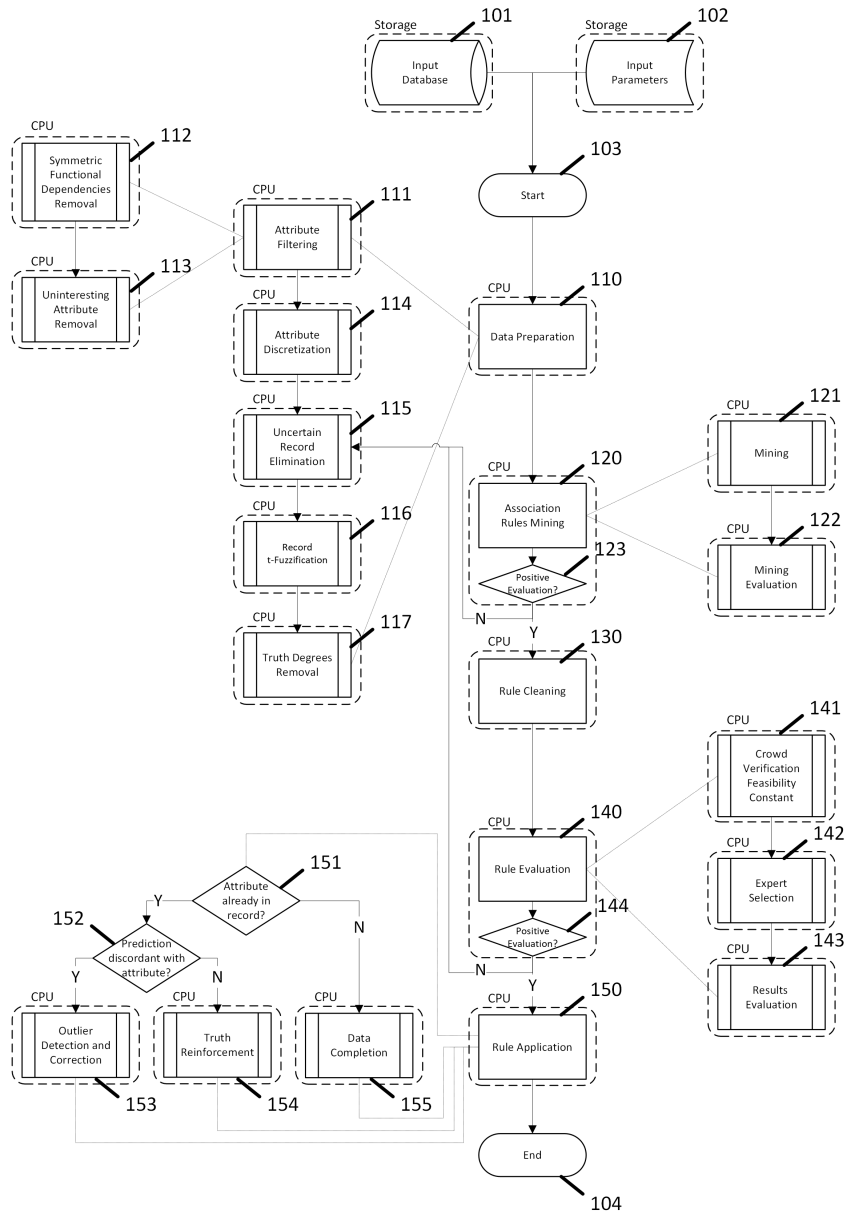


Figure 5.1: The figure presents an overview of all the components constituting $ARM(D, I)$. Dashed lines are used to indicate the mapping of a software element to hardware components. Dotted lines are used as a magnifier that shows the subprocesses of a process.

$\sigma_i \in \mathbb{N}$, then σ_i indicates the number of classes into which the user intends to divide the values of attribute $b_i \in B$. If $\sigma_i = \text{true}$, it indicates performing an unsupervised segmentation method for the attribute b_i . If $\sigma_i \in MF$, then the value $q \in \sigma_i = \{mf, q\}$ indicates in how many classes the values will be discretized using the membership function mf . Finally, if σ_i is not provided, a default value of $\sigma_i = 5$ is used.

5. Parameter $m \in [0, 1]$, where, in case non-crisp truth values are provided together with data values in A , m is a threshold value indicating that any record r in which at least one truth value $t(a, r) < m$, for each $a \in A$, should not be considered for the Association Rules Mining (**120**). Thus, r should not be part of the training data. (However, such tuples will be considered in the process of Rule Application (**150**) described further below.) If not provided, a default value of $m = 0.5$ is used.
6. Parameters $\langle \tau, t \rangle$. Here, $\tau \in \{L, P, G\}$ specifies how the aggregate truth value t_r of an entire record $r \in D$ is computed using the truth values of its attributes, whereas t states the minimum truth value for each record to be considered. In particular, τ refers to the t-norm used in the Record t-fuzzification process described further below. This can be one of $\{\text{Łukasiewicz t-norm, Product t-norm, Gödel t-norm}\}$, indicated by their initials $\{L, P, G\}$. Each record r for which $t_r < t$ should be disregarded by the rule mining process (for example, by eliminating such records from D). If not provided, fixed default values of $\tau = L$ and $t = 0.5$ are used.
7. Parameter $S \in (0, 1]$ indicating the minimum support cutoff constant expressed as a fraction. As will be explained later, this constant is used to eliminate mined itemsets with low support. If not provided, a default value of $S = 0.05$ is used.
8. Parameter $C \in (0, 1]$ indicating the minimum confidence cutoff constant expressed as a fraction. As explained later, this constant is used to eliminate mined itemsets with low confidence. If not provided, a default value of $C = 0.75$ is used.
9. Parameter nr_{min} indicating the minimum number of rules to be mined, where $nr_{max} \geq nr_{min}$. If not provided, a default value of $nr_{min} = 10$ is used.
10. Parameter nr_{max} indicating the maximum number of rules to be mined, where $nr_{max} \geq nr_{min}$. If not provided, a default value of $nr_{max} = 1000$ is used.

11. Parameter $K \in (0, 1]$ indicating the minimum Kulczynski measure value. If not provided, a default value of $K = 0.75$ is used.
12. Parameter $KIR \in (0, 1]$ indicating the minimum Kulczynski and Imbalance Ratio combination value. If not provided, a default value of 0.75 is used.
13. Parameter $ARM \in \{FP, Apriori, Eclat\}$, where ARM is a standard association rule mining algorithm chosen from the set $\{FP - growth, Apriori, Eclat\}$. If not provided, a default value of FP is used.
14. Parameter $l \in [0, 1]$, where l is the fraction of rules to be tested to estimate the overall quality of the rule set. If not provided, a default value of $l = 0.3$ is used.
15. Parameter $HITL \in \mathbb{N}$ indicating the Human In The Loop throughput capacity. $HITL$ is the number of tasks that can be elaborated by the human validators involved in the verification of the process. If not provided, a default value of $HITL = 0$ is used.
16. Parameter $p \in [0, 1]$ indicating the minimum accuracy fraction desired from the evaluation of the generated rules. If not provided, a default value of $p = 0.70$ is used.
17. Structure $SS = \langle ss_m, ss_t, ss_S, ss_C, ss_{nr_{min}}, ss_{nr_{max}}, ss_K, ss_{KIR}, ss_l, ss_p \rangle$ where each element of S indicates the step size expressed as a fraction to be used if the procedure needs to trade the overall quality of the results produces for termination guarantees of the overall process. If not provided, a default structure $SS = \langle ss_m = 0.1, ss_t = 0.1, ss_S = 0.1, ss_C = 0.1, ss_{nr_{min}} = 0.1, ss_{nr_{max}} = 0.1, ss_K = 0.1, ss_{KIR} = 0.1, ss_l = 0.1, ss_p = 0.1 \rangle$ is used.
18. Parameter $e_b \in [0, 1]$ indicating the relative importance used to account for revisions of attributes performed by an expert in the body of a rule. If not provided, a default value of $e_b = 0.25$ is used.
19. Parameter $e_h \in [0, 1]$ indicating the relative importance used to account for attributes' revisions performed by an expert in the head of a rule. If not provided, a default value of $e_h = 1$ is used.
20. Parameter $TO \in \mathbb{N}$ indicating the timeout, expressed in minutes, for receiving expert feedback. If not provided, a default value of $TO = 180$ is used.

21. Parameter $tr_{old} \in [0, 1]$ indicating the relative importance that the truth value of a record should hold when performing the update of the truth degrees during the Truth Reinforcement **(154)** process described further below. If not provided, a default value of $tr_{old} = 0$ is used.
22. Parameter $tr_{new} \in [0, 1]$ indicating the relative importance that the confidence of an association rule should hold when performing the update of the truth degrees of the records in D_0 **(101)** during the Truth Reinforcement **(154)** process described further below. If not provided, a default value of $tr_{new} = 1$ will be used.

5.1.2 Data Preparation (110)

The relation schema A of the input database D might contain redundant and/or uninteresting attributes for mining association rules. Using these attributes would lead to an unnecessarily slower Mining **(121)** process due to the necessity to consider larger itemsets/FP-trees (depending on the utilised ARM algorithm). Therefore, excluding such attributes from the data considered in the procedure is preferable. Moreover, trying to mine association rules whose attributes have a domain with a wide range of values can slow down the procedure. Therefore, the current process executes a set of sub-processes to prepare the input data by eliminating (projecting out) unnecessary attributes.

Moreover, the system uses novel methodologies to handle fuzzy values for the attributes of records in D . The problem of mining association rules in such a setting is addressed by making use of a mixture of t-normalization techniques and thresholding, guided by the specifications given in the input parameters I **(102)**. Here below, an example database D_{ex} is introduced for showcasing the effect of the application of the various sub-processes of the Data Preparation **(110)**.

<i>id</i>	<i>name</i>	<i>description</i>	<i>#employees</i>	<i>income</i>	<i>esg</i>	<i>t_{esg}</i>	<i>sent</i>	<i>t_{sent}</i>
1	Alphabet	Alphabet is ...	150K	80B	medium	0.8	+	1
2	Apple	Apple is ...	130K	100B	high	1	=	0.8
3	Meta	Meta is ...	70K	120B	medium	0.3	=	0.5
4	Microsoft	Microsoft is ...	22K	60B	low	0.9	+	0.7
5	Tesla	Tesla is ...	110K	5B	high	0.2	-	0.7
6	ACME	ACME is ...	100K	20B	low	0.7	-	1

Table 5.1: Example database D_{ex} over company data including fuzzy values for two attribute.

The database D_{ex} contains nine columns, 2 of which represent truth values. $id \in \mathbb{N}$ is the primary key of the database. $name$ and $description$ are strings representing, respectively, the name and description of the companies. $\#employees \in \mathbb{N}$ represents the number of employees in a company. $esg \in \{\text{low}, \text{medium}, \text{high}\}$ and $sent \in \{+, -, =\}$ are scores representing, respectively, the environmental, social, and governance (esg) score of the company and an analysis of the clients' sentiment toward the company. Finally, $t_{esg} \in [0, 1]$ and $t_{sent} \in [0, 1]$ represent, respectively, the truth values of the columns esg and $sent$. The sub-processes of Data Preparation (110) are now described in order.

Attribute Filtering (111)

The process "Attribute Filtering", mentioned in Figures 5.1 and A.2, safely eliminates attributes of two classes from the database D and the corresponding relation schema A . These attributes correspond to non-informative data concerning the scope of finding association rules. This process is essential to improve the performance of the Mining (121) by reducing the quantity of data to analyze and simplifying the resulting rules, generating more concise rule bodies.

The Attribute Filtering subprocess is composed of two further subprocesses: Symmetric Functional Dependencies Removal (112) and Uninteresting Attribute Removal (113). Both are explained in detail further below.

Symmetric Functional Dependencies Removal (112)

The process "Symmetric Functional Dependencies Removal", mentioned in Figures 5.1 and A.3, in case the set FD of functional dependencies in A is not empty, uses standard algorithms to iteratively find all symmetric pairs of FDs $a \rightarrow a'$ and $a' \rightarrow a$, for (single) attributes $a, a' \subseteq A$. After each iteration, if $a' \notin B$, eliminate a' from A (that is: $A := A \setminus \{a'\}$), otherwise eliminate a from A (i.e., $A := A \setminus \{a\}$). If either $a \in U$ or $a' \in U$, delete both a and a' . However, the set FD of FDs remains unchanged.

For example, if the input I (102) specifies $FD = \{id \rightarrow name, name \rightarrow id, name \rightarrow description, description \rightarrow name\}$, $U = \{description\}$, the result of the Symmetric Functional Dependencies Removal (112) on D_{ex} is as shown in Table 5.2, where all the attributes appearing in the symmetric functional dependencies have been removed since $description \in U$ is part of the series of symmetric functional dependencies to be removed.

<i>#employees</i>	<i>income</i>	<i>esg</i>	<i>t_{esg}</i>	<i>sent</i>	<i>t_{sent}</i>
150K	80B	medium	0.8	+	1
130K	100B	high	1	=	0.8
70K	120B	medium	0.3	=	0.5
22K	60B	low	0.9	+	0.7
110K	5B	high	0.2	-	0.7
100K	20B	low	0.7	-	1

Table 5.2: Example database D_{ex} after the removal of symmetric functional dependencies.

Uninteresting Attribute Removal (113)

The process "Uninteresting Attribute Removal", mentioned in Figures 5.1 and A.3, removes from A all the uninteresting attributes U indicated in the input I (102), such that $A' = A \setminus U$. For the rest of the procedure, retain only the attributes in A' . For example, if I specifies $U = \{description\}$, the result of the Uninteresting Attribute Removal (113) on D_{ex} is the same as 5.2, since *description* is part of a series of symmetric functional dependencies in FD .

Attribute Discretization (114)

The process "Attribute Discretization", mentioned in Figures 5.1 and A.2, refers to the process of mapping and transferring continuous or very large domains (or ranges) of certain attributes into discrete smaller counterparts, and replacing data values of these attributes in the database D accordingly. Thus, by abuse of language, discretization here does not only apply (as usual) to *infinite* domains (such as the integers or the rational numbers). It also applies to *large* domains or sets of values, that smaller discrete domains of new values will replace, each new value usually corresponding to a class or interval of old values.

Association rules find their historical origin in the Market Basket Analysis problem, where the value to predict is either true or false (i.e. the item is in or not in the basket). The association rules were originally not meant to predict that an attribute value belongs to a certain class among an arbitrarily high number of classes. To produce association rules that take into account attributes whose domain values' range is infinite (or, anyway, vast), these attributes must be discretized into a small finite number of classes representing intervals of values. For each $b[\sigma] \in B$ provided in I (102), the procedure performs a customized discretization of the attribute b according to the method specifier σ :

- If $\sigma \in \mathbb{N}$, the values of the column $b \in D$ are ordered in an ascendant fashion, considering either a numerical order in case the column type is numerical, or a lexicographical order otherwise. Computing the cardinality of the active domain $|AD_b|$, where the active domain of a column is the set of values appearing in the column itself, the values on b are divided into σ segments of approximately $\frac{|AD_b|}{\sigma}$ values. Each record value v in the column b is replaced with a string $c \in \{C_1, \dots, C_\sigma\}$ representing the membership to the computed class.
- If $\sigma \in MF$, the values of the column $b \in D$ are discretized into q classes identified by the membership function mf , where $\sigma = \{mf, q\}$. The mf is replicated q times and located equidistantly into the active domain AD_b . Each record value for the attribute b is replaced with a string $c \in \{C_1, \dots, C_q\}$ indicating the membership to the nearest mf .
- If $\sigma = true$, Jenks Natural Breaks Optimization [92] is performed to find a natural distribution for the values of the column $b \in D$. Each record value for the attribute b is replaced with a string $c \in \{C_1, \dots, C_j\}$, where j is the number of classes individuated by the algorithm.

For example, if I specifies $B = \{\#employees[3], income[3]\}$, the result of the Attribute Discretization (114) on D_{ex} is:

<i>#employees</i>	<i>income</i>	<i>esg</i>	<i>t_{esg}</i>	<i>sent</i>	<i>t_{sent}</i>
C_3	C_2	medium	0.8	+	0.7
C_3	C_3	high	1	=	0.8
C_1	C_3	medium	0.3	=	0.5
C_1	C_2	low	0.9	+	0.7
C_2	C_1	high	0.2	-	0.7
C_2	C_1	low	0.7	-	1

Table 5.3: Example database D_{ex} after the attributes discretization.

Uncertain Record Elimination (115)

The process "Uncertain Record Elimination", mentioned in Figures 5.1 and A.2, serves to delete records not meeting minimum truth levels. For each record $r \in D$, if any attribute $a \in A$ has a truth value $t(r, a) < m$, where m is the minimum truth value for each attribute in a record to be considered valid indicated in I (102), exclude r_i from the procedure. This ensures that no record containing an

extremely low truth value on any of its attributes influences the association rules generated. In case $m = 0$, no record is eliminated in this process. For example, if I specifies $m = 0.3$, then the result of the Uncertain Record Elimination (115) on D_{ex} is:

#employees	income	esg	t_{esg}	sent	t_{sent}
C_3	C_2	medium	0.8	+	0.7
C_3	C_3	high	1	=	0.8
C_1	C_3	medium	0.3	=	0.5
C_1	C_2	low	0.9	+	0.7
C_2	C_1	low	0.7	-	1

Table 5.4: Example database D_{ex} after the removal of uncertain records.

Record t-Fuzzification (116)

The process "Record t-Fuzzification", mentioned in Figures 5.1 and A.2, enforces minimum truth levels. For each record $r \in D$, calculate the truth value $t(r)$ of r using the t-norm τ specified in I . If $t(r) < t$ delete r from D otherwise store $t(r)$ to recall it later.

This ensures that the aggregated truth value of each record utilised in the Mining (121) process of association rules will have minimum truth guarantees. For example, if I specifies that $\tau = L$ and $t = 0.5$, then the result of the Record t-Fuzzification (116) on D_{ex} is:

#employees	income	esg	t_{esg}	sent	t_{sent}	t_{record}
C_3	C_2	medium	0.8	+	0.7	0.5
C_1	C_2	low	0.9	+	0.7	0.6
C_2	C_1	low	0.7	-	1	0.7

Table 5.5: Example database D_{ex} after the Record t-Fuzzification (116).

Truth Degrees Removal (117)

The process "Truth Degrees Removal", mentioned in Figures 5.1 and A.2, removes all the attributes t_a and t_r indicating respectively the truth values of the attribute a or the truth value of the records in D . The truth values will not be used by subsequent processes in the algorithm. For example, if the schema $A = (\#employees, income, esg, t_{esg}, sent, t_{sent}, t_{record})$, then the result of the Truth Degrees Removal (117) on D_{ex} is:

<i>#employees</i>	<i>income</i>	<i>esg</i>	<i>sent</i>
C_3	C_2	medium	+
C_1	C_2	low	+
C_2	C_1	low	–

Table 5.6: Example database D_{ex} after the removal of truth degrees from the database.

5.1.3 Examples of Variations of Data Preparation (110)

Here, we present a list of possible variations that can be applied to the process of Data Preparation (110).

Examples of Alternative Symmetric Functional Dependencies Removal (112)

The method can dynamically check multiple times whether some $FDs\ a \rightarrow a'$ follow from FD . Instead, an example may first compute by well-known methods the set FD^+ of all non-trivial FDs that logically follow from FD , and then in order to check $a \rightarrow a'$, merely test whether $a \rightarrow a'$ is contained in FD^+ .

Alternatively, if the set FD of functional dependencies in A is not provided, then utilise standard algorithms for the discovery of FDs and proceed as usual for the removal of symmetric functional dependencies.

Examples of Alternative Attribute Discretization (114)

For each $b[\sigma] \in B$ provided in I (102), the procedure performs a customized discretization of the attribute b according to the method specifier σ as described in Attribute Discretization (114). A further parameter $min_truth_discretization \in [0, 1]$ is required in I (102). $min_truth_discretization$ indicates the minimum truth degree that discretized attributes are going to retain at the marginal values of the segments identifying different classes. While performing the Attribute Discretization (114), a truth value for each attribute b is retained in a possibly new column of $t_b \in D$. In case t_b was not already present in D , it is computed as:

- If $\sigma \in \mathbb{N}$ and $\frac{|AD_b|}{\sigma}$ is the cardinality of values in a segment, for each segment c_i , with $i \in [2, \dots, \sigma - 1]$, store a truth value $t(b, r) = 1$ in the column t_b for the record r if $val(b, r)$ lies in the middle of the segment. Otherwise, gradually discount $t(b, r)$ up to the threshold $min_truth_discretization$ for

values $val(b, r)$ that are on the edge of the segment. For segments c_1 and c_σ this is applied only on the second half of segment 1 and the first half of segment σ , whereas for the first half of segment 1 and the second half of segment σ there is no discounting.

- If $\sigma \in MF$, where $\sigma = \{mf, q\}$, the truth values for t_b are automatically identified upon the use of the membership function mf .
- If $\sigma = true$, Jenks Natural Breaks Optimization is performed and a similar behaviour as when $\sigma \in \mathbb{N}$ is applied.

In case t_b was already present in A , meaning that the attribute b already has a fuzzy value associated, then compute t'_b as described above and overwrite the values in t_b as $val(t_b, r) = val(t_b, r) \odot_L val(t'_b, r)$, where $\odot_L$ is the standard Łukasiewicz t-norm.

As an additional alternative, if $\sigma \in \mathbb{N}$, the class size used for splitting the records in D into different classes can be computed as $\frac{\max(b) - \min(b)}{\sigma}$.

A further variation might include methods of adjusting the truth value associated with the discretized attributes. In some applications, mapping the value of an attribute of a record $v(a, r)$ into a restricted domain might influence the truth value $v(t_a, r)$. Assume, for example, that the salary of all managers of a company ACME is known and asserted with truth value 1 in the database before discretization. These salaries are then discretized into four category-values: "low" for all salaries that are less than 100,000, "medium" for salaries in the (right-open) interval $[100000, 200000[$, "high" for salaries in the interval $]200000, 500000]$ and "very high" corresponding to all salaries greater than 500000. Now, for example, it could be meaningful to lower the original truth values for discretized attribute values whose original (non-discretized) values lie at the border of the relevant interval. For example, assume that Marta's salary was 500001 in the original database with truth value 1. In the discretized database this could then be "very high" with a truth value of 0.9, as this value is close to the border between "high" and "very high". Such adjustments are really application-dependent and could be expressed through an application-specific truth-value adjustment function taf_a for an attribute $a \in A$ to be discretized. For each original a -value $v(a, r)$ for some record r , with associated certainty $v(t_a, r)$ the function application $taf_a(v(a, r), v(t_a, r))$ will yield the truth value $v(t_a, r)$ for the updated record r after discretization. For a number of attributes a , the associated functions taf_a may be specified as part of the input parameters I .

When $\sigma \in MF$, the membership functions themselves are a natural choice for possible concrete incorporations of the functions taf .

Examples of Alternative of Record t-Fuzzification (116)

To perform the Alternative Record t-Fuzzification, a new parameter $mul \in \mathbb{N}$ is required in I (102). If not provided, a default value of $mul = 5$ will be used. Compute the aggregated truth value t_{r_i} of each record $r_i \in D$ using the t-norm τ specified in the input I and the multiplier mul specified in I and store it together with r_i . Construct a new database D' where each record $r_i \in D$ is written a number of times equal to $round(t_{r_i} * mul)$, where the function $round()$ takes in input a float number and returns the nearest integer. This will create a larger database which will reflect the record-relative significance through the number of times that the record appears in the database. This behaviour, during the Mining (121) of association rules, pushes down the confidence and support values of records with smaller truth values. For example, if the input I specifies that $\tau = L$ and $mul = 5$, the result of the Alternative Record t-Fuzzification on D_{ex} is:

#employees	income	esg	t_{esg}	sent	t_{sent}	t_{record}
C_3	C_2	medium	0.8	+	0.7	0.5
C_3	C_2	medium	0.8	+	0.7	0.5
C_3	C_2	medium	0.8	+	0.7	0.5
C_3	C_3	high	1	=	0.8	0.2
C_1	C_2	low	0.9	+	0.7	0.6
C_1	C_2	low	0.9	+	0.7	0.6
C_1	C_2	low	0.9	+	0.7	0.6
C_2	C_1	low	0.7	–	1	0.7
C_2	C_1	low	0.7	–	1	0.7
C_2	C_1	low	0.7	–	1	0.7
C_2	C_1	low	0.7	–	1	0.7

Table 5.7: Example database D_{ex} from Table 5.4 after the Alternative Record t-Fuzzification.

Examples of Alternative of Truth Degree Removal (117)

Remove the attribute t_r indicating the aggregated truth value of the records in D . Note that the attributes t_a , for each $a \in A$, are preserved to be used in Alternative processes of the procedure.

For example, if the schema $A = (\#employees, income, esg, t_{esg}, sent, t_{sent}, t_{record})$, then the result of the Truth Degrees Removal (117) on D_{ex} is:

$\#employees$	$income$	esg	t_{esg}	$sent$	t_{sent}
C_3	C_2	medium	0.8	+	0.7
C_1	C_2	low	0.9	+	0.7
C_2	C_1	low	0.7	–	1

Table 5.8: Example database D_{ex} after the removal of the record aggregated truth value from the database.

5.1.4 Association Rules Mining (120)

Mining (121)

The process "Mining", mentioned in Figures 5.1 and A.4, performs over D the Mining (121) task using a pre-built implementation of the selected algorithm ARM indicated in I (102), with confidence and support minimum fraction values of, respectively, C and S indicated in I . This produces as output a list of association rules AR of the form:

$$body \rightarrow head \quad (5.1)$$

where $body$ is a non-empty list (expressing a conjunction in case of more than two elements) of one or more ground atoms, and $head$ is a single ground atom. The procedure also uses the set Int of known interesting attributes to be used as targets of the association rules predictions, i.e. as the ground atom in the $head$ of each ar .

The Mining (121) process and the selected support framework generate association rules as discussed below. In particular, the case where $ARM = Apriori$ is outlined, where the constant $Apriori$ indicates the use of the standard Apriori association rules mining algorithm. It employs an iterative approach to compute all the frequent itemsets, where a frequent itemset is defined as the set of attribute values that appear frequently in D . It starts by computing all the frequent itemsets of dimension 1 and, in each iteration, uses all the itemsets of iteration $k - 1$ to compute the itemsets of iteration k . The algorithm computes a support value associated with each candidate itemset $AV = \{a_1(v_1), \dots, a_k(v_k)\}$, where, for $1 \leq i \leq k$, $a_i \in A$ and $v_i = val(a_i, r)$ for some record r of D , computed as:

$$S(AV) = P(AV) = \frac{\text{number of records containing } AV}{\text{number of all records in } D} \quad (5.2)$$

All the candidate itemsets of a given iteration k satisfying the minimum support fraction S indicated in I are consolidated as itemsets of the iteration k . To compute the itemsets of iteration k , a join step between all of the itemsets at iteration $k - 1$ is performed, verifying the satisfaction of the minimum support for the new candidate itemsets.

Once all the frequent itemsets are generated, these are used to generate association rules, using one of the attribute values in the itemset as the *head* of the rule and the remaining attribute values as the *body* of the rule. The set of attributes Int provided in I is used to reduce the amount of generated rules. If $Int \neq \emptyset$, the attribute values used in *head* are drawn only from the attributes in Int .

Consider an association rule ar of the form shown in Equation 5.1, where $head = a(v)$. Then, for each record r , *body* represents the set of conditions that have to be satisfied to predict that the value of the attribute a of r is v .

To filter out weak association rules, the algorithm *ARM* associates a confidence value $C(ar)$ to each computed association rule $ar \in AR$ as follows:

$$C(ar) = P(head(ar)|body(ar)) = \frac{S(ar)}{S(body(ar))} \quad (5.3)$$

where $S(ar) = S(body(ar) \cup head(ar))$. Each $ar \in AR$ will respect the minimum value of $C(ar) > C$ and $S(ar) > S$, where C and S are provided in the input I .

If the database used to perform the Mining (121) process is D_{ex} described in Table 5.6, the input I specifies $Int = esg, sentiment, income(C)$, assuming a more numerous set of tuples used for the Mining (121), a plausible set of association rules would be:

$$\begin{aligned} \#employees(C_i), income(C_j) &\rightarrow esg(low) \\ \#employees(C_m), income(C_n) &\rightarrow sentiment(+) \\ esg(high) &\rightarrow income(C_l) \\ &\dots \end{aligned}$$

Here above, the classes of *#employees* and *income* use generic indexes to keep the example more abstract.

Mining Evaluation (122)

The process "Mining Evaluation", mentioned in Figures 5.1 and A.4, serves to evaluate the results of the Mining process. If the number of produced association

rules $|AR| < nr_{min}$ or $|AR| > nr_{max}$, where nr_{min} and nr_{max} are, respectively, the desired minimum and maximum number of rules indicated in the input I (102), then the evaluation is not considered successful (123). The computation restarts from Uncertain Record Elimination (115). In the case of $|AR| < nr_{min}$, a Parameter Modification is performed over the input parameters as follows:

$$\begin{aligned} m &:= m - (m * ss_m) \\ t &:= t - (t * ss_t) \\ S &:= S - (S * ss_S) \\ C &:= C - (C * ss_C) \\ nr_{min} &:= nr_{min} - (nr_{min} * ss_{nr_{min}}) \end{aligned}$$

In the case of $|AR| > nr_{max}$, the above-presented Parameter Modification is modified to update the input parameters as follows:

$$\begin{aligned} m &:= m + (m * ss_m) \\ t &:= t + (t * ss_t) \\ S &:= S + (S * ss_S) \\ C &:= C + (C * ss_C) \\ nr_{max} &:= nr_{max} + (nr_{max} * ss_{nr_{max}}) \end{aligned}$$

Intuitively, the minimum (maximum) threshold is relaxed to facilitate the procedure to satisfy it while, at the same time, other more relaxed (tightened) parameters are enforced to produce more (fewer) rules. This converges with the number of mined rules gradually increasing (decreasing) and the threshold on the number of rules gradually decreasing (increasing). By tuning the step size input parameters, it is possible to allow for a faster (or slower) convergence. Moreover, imposing non-zero step size parameters also allows one to specify which input parameters will be affected by the relaxation.

If the number of association rules produced $nr_{max} \leq |AR| \leq nr_{min}$, then the evaluation is considered successful (123) and the computation continues with the Rule Cleaning (130).

5.1.5 Examples of Variations of Association Rules Mining (120)

Here, we present a list of possible variations that can be applied to the process of Association Rules Mining (120).

Examples of Alternative of Mining (121)

Instead of using the classical notions for the values of support and confidence utilised during the mining of association rules, novel notions of fuzzy support and fuzzy confidence are here introduced and employed during the mining process.

To make use of the fuzzy extension of the association rule mining algorithms, the variation of Truth Degrees Removal must be used to preserve the truth values associated with specific attributes.

During the candidate frequent itemset generation phase, for each generated non-empty candidate itemset $AV = \{a_1(v_1), \dots, a_k(v_k)\}$, where, for $1 \leq i \leq k$, $a_i \in A$ and a_i is not a truth-value attribute, compute the support of AV as:

$$S(AV) = \frac{\sum_{r \in D, AV \subseteq r} (val(t_{a_1}, r) \odot \dots \odot val(t_{a_k}, r))}{\text{number of all records in } D} \quad (5.4)$$

where $\odot$ is the t-norm specified in $M \in I$, $AV \subseteq r$ indicates that each element in the itemset AV is present in the tuple r , and $val(t_{a_j}, r) = 1$ where t_{a_j} is not specified for r .

Examples of Alternative of Mining Evaluation (122)

Instead of relaxing both the minimum number of rules nr_{min} (or the maximum number of rules nr_{max}) and the other relevant parameters (m , M , S and C), relax only some of these parameters, or relax all but nr_{min} and nr_{max} .

Another variation may require four additional parameters m_{min} , M_{min} , S_{min} , C_{min} to be provided as part of the input I . Each of these parameters indicates a critical minimum threshold below which the parameters can not be lowered during the relaxation of parameters. This prevents borderline cases where meaningless rules would be produced after relaxations.

5.1.6 Rule Cleaning (130)

The process "Rule Cleaning", mentioned in Figures 5.1 and A.1, enforces advanced constraints to the rules produced. Metrics such as support and confidence might be misleading as they fail to measure the real strength of the correlation and implication between the *body* and the *head* of an association rule [84]. Therefore, the procedure also uses other metrics such as the Kulczynski measure (Kulc),

calculated for each association rule ar as:

$$Kulc(ar) = \frac{1}{2} \left(P(body(ar) \mid head(ar)) + P(head(ar) \mid body(ar)) \right) \quad (5.5)$$

and the Imbalance Ratio (IR), calculated as:

$$IR(ar) = \frac{|S(body(ar)) - S(head(ar))|}{S(body(ar)) + S(head(ar)) - S(body(ar) \cup head(ar))} \quad (5.6)$$

to remove statistically irrelevant rules. If the two directional implications, i.e. $body(ar)$ implies $head(ar)$ and vice versa, have the same strength, then $IR(ar)$ will be zero. Here, the strength of an implication is measured by how many times the implication is true in the database D . Otherwise, the larger the difference between the two directional implications, the larger the imbalance ratio. The Kulczynski measure represents the average of two conditional probabilities: the probability of itemset $head(ar)$ given itemset $body(ar)$, and the probability of itemset $body(ar)$ given itemset $head(ar)$. The conditional probabilities can be viewed as a confidence measure for the rule. The Imbalance Ratio is *null-invariant* as it is not affected by the number of null transactions and is independent of the total number of transactions. Let K and KIR be the minimum thresholds specified in I for the Kulczynski measure and the combination of the Kulczynski measure with the Imbalance Ratio, respectively. The best rules are those where either (i) $Kulc(ar) > K$ and is thus close to 1 or (ii) $Kulc(ar) \leq K$ and $2 * |0.5 - Kulc(ar)| * IR(ar) > KIR$. Note that the Rule Cleaning (130) carried out during this process is exemplary while different formulas for the combination of $Kulc$ and IR can be devised. These different formulas would lead to minor variants, several of which can be meaningful in practice.

5.1.7 Examples of Variations of Rule Cleaning (130)

Here, we present a list of possible variations that can be applied to the process of Rule Cleaning (130).

Instead of using the preferred metrics $Kulc$ and IR , another example may include receiving as part of the input I (102) a set of functions together with the expected min or max value that each function should return when applied to an association rule and a database. The set of functions would be of the form $F = \{\langle f_1(ar, D), min/max \rangle, \dots, \langle f_n(ar, D), min/max \rangle\}$, where min/max stands for an equation such as $min = 0.1$ or $max = 0.9$. Here some examples of evaluation functions (i.e. metrics) that can be used to score the quality of association rules are provided:

1. *lift*, computed as $lift(body, head) = \frac{P(body \cup head)}{P(body) * P(head)}$. A value of *lift* < 1 indicates that the *body* is *negatively correlated* with the *head* of the rule, where a pair of set of attributes is said to be *negatively correlated* if the presence of one set implies that the other set will likely be absent. On the contrary, a value of *lift* > 1 indicates a *positive correlation*. Finally, a value of *lift* = 1 indicates an absence of correlation.
2. χ^2 , which makes use of a contingency table with observed and expected values that can be generated with standard techniques. χ^2 can then be computed as $\sum \frac{(observed - expected)^2}{expected}$.
3. *all confidence*, computed as $\min\{P(body|head), P(head|body)\}$. It represents the minimum confidence of the two association rules $body \rightarrow head$ and $head \rightarrow body$. *all confidence* is influenced by the support of the *body*, the *head* and $body \cup head$ but not by the total number of records.
4. *max confidence*, computed as $\max\{P(body|head), P(head|body)\}$. It represents the maximum confidence of the two association rules $body \rightarrow head$ and $head \rightarrow body$. *max confidence* is influenced by the support of the *body*, the *head* and $body \cup head$ but not by the total number of records.
5. *cosine measure*, computed as $\sqrt{P(body|head) * P(head|body)}$. It represents a harmonized lift value, that is influenced by the support of the *body*, the *head* and $body \cup head$ but not by the total number of records.

Note that the presented list is an incomplete list of metrics that can be used to measure the statistical relevance of the produced association rules.

5.1.8 Rule Evaluation (140)

Crowd Verification Feasibility Constant (141)

The process "Rule Evaluation", mentioned in Figures 5.1 and A.5, calculates the amount *cvf* of association rules from *AR* to be evaluated by the (crowd of) domain experts involved in the evaluation of $ARM(D, I)$. The throughput capacity that can be handled by the human component is indicated in the input parameter *HITL* (Human In The Loop).

If $HITL < |AR| * l$ then set $cvf = HITL$. If $|AR| \geq HITL \geq |AR| * l$ then set $cvf = |AR| * l$. Otherwise, if $HITL \geq |AR|$ then set $cvf = |AR|$.

Randomly select a set $G \subseteq AR$ of cardinality *cvf* to be sent to the crowd.

Expert Selection (142)

Each time that an expert e reviews an association rule $ar \in G$ of the form $a_1(v_1), \dots, a_{n-1}(v_{n-1}) \rightarrow a_n(v_n)$, update E as the following:

1. for each $a_i(v_i) \in ar$, if the tuple $(e, a_i, rev) \in D_e$ then update it as $(e, a_i, rev + e_b)$, where $1 \leq i \leq n - 1$ and $e_b \in I$ is the relative importance assigned to revisions of body attributes of a rule. If $(e, a_i, rev) \notin D_e$, add (e, a_i, e_b) to E .
2. for $a_n(v_n) \in ar$, if the tuple $(e, a_i, rev) \in D_e$ then update it as $(e, a_i, rev + e_h)$, where $e_h \in I$ is the relative importance assigned to revisions of head attributes of a rule. If $(e, a_i, rev) \notin D_e$, add (e, a_i, e_h) to E .

In the following, an example of the database E is provided:

<i>expert_id</i>	<i>attribute</i>	<i>#reviews</i>
123456	<i>#employees</i>	20
123456	<i>income</i>	5
123457	<i>income</i>	30

Table 5.9: Example database E containing information regarding the reviews performed by experts.

For each association rule $ar \in G$ and for each expert e identified by eid , compute a score s as follows:

$$s(eid, ar) = \sum_{\substack{(eid, attr, rev) \in E, \\ attr \in ar}} rev \quad (5.7)$$

Assign each $ar \in G$ to an available expert e for evaluating ar based on domain expert knowledge, where e is identified by an expert identifier eid such that $s(eid, ar)$ takes a maximum value when eid ranges over all available experts. If more than one such expert has the maximum value $s(eid, ar)$, then choose one among them randomly. Moreover, if no ranking of the experts can be computed due to lack of data in E , then assign each $ar \in G$ randomly. Considering the example database containing information regarding experts' reviews shown in Table 5.9, it is clear that the expert with associated id "123457" would be chosen for evaluating rules involving the attribute *#employees*, whereas the expert with associated id "123456" would be the preferred choice for rules involving the attribute *income*.

The system sends to each expert e the rules that have been assigned to e and awaits the rule evaluation feedback from e . If some e does not provide feedback within

the timeout $TO \in I$, then e is marked unavailable, and a new expert assignment for the assigned association rule is computed by the above procedure. Each expert e , for each association rule ar assigned to e , will provide feedback in a message of the form: $\langle e, ar, feed \rangle$ where $feed \in \{true, false\}$. In this setting, $feed = true$ indicates positive feedback for the rule ar and $feed = false$ indicates negative feedback, meaning that the rule is not correct according to e 's domain knowledge and their web searches.

Results Evaluation (143)

Once the feedback for each rule ar from the previous process has been received, the system aggregates them and computes the overall accuracy of the rule set G as follows:

$$accur(G) = \frac{\text{number of positive expert validations of rules in } G}{\text{number of all expert validations of rules in } G} \quad (5.8)$$

If $accur(G) < p$, where p is the desired minimum accuracy fraction indicated in I (102), then the evaluation is not considered successful (144). The computation restarts from the Uncertain Record Elimination (115) after performing a Parameter Modification over the input parameters as follows:

$$\begin{aligned} m &:= m + (m * ss_m) \\ t &:= t + (t * ss_t) \\ S &:= S + (S * ss_S) \\ C &:= C + (C * ss_C) \\ nr_{min} &:= nr_{min} - (nr_{min} * ss_{nr_{min}}) \\ K &:= K + (K * ss_K) \\ KIR &:= KIR + (C * ss_{KIR}) \\ p &:= p - (p * ss_p) \end{aligned}$$

where each $ss_i \in SS$ is defined in I .

Otherwise, if $accur(G) \geq p$ the evaluation is considered successful (144) and the system continues its computations with Rule Application (150) as described further below.

5.1.9 Examples of Variations of Rule Evaluation (140)

Here, we present a list of possible variations that can be applied to the process of Rule Evaluation (140).

Examples of Alternative of Crowd Verification Feasibility Constant (141)

A variation includes sending to an expert, together with an association rule ar , a non-empty list of records R_{ar} , where each record in R_{ar} satisfies the conditions stated in $body(ar)$ but violates the condition in $head(ar) = a(v)$. Such a violation arises either because there is no value $val(a, r)$ (for example, in a strictly relational database, indicated by the presence of a null value such as "NULL"), or because there is a value $val(a, r) \neq a(v)$. This allows the experts to verify, for example through web searches, both the general validity of the rule and the specific application of the rule to real cases. Given the input parameters $HITL$ and l , compute the amount cvf of association rules from AR that can be evaluated by the domain experts involved and the maximum cardinality of R_{ar} as (i) if $HITL < |AR| * l$ then set $cvf = HITL$ and $|R_{ar}| = 1$, (ii) if $|AR| \geq HITL \geq |AR| * l$ then set $cvf = |AR| * l$ and $|R_{ar}| = \max \left\{ \frac{HITL - cvf}{cvf}, 1 \right\}$, otherwise (iii) if $HITL \geq |AR|$ then set $cvf = |AR|$ and $|R_{ar}| = \max \left\{ \frac{HITL - cvf}{cvf}, 1 \right\}$. Randomly select a set $G \subseteq AR$ of cardinality cvf . For each $ar \in G$, select from D an amount of $|R_{ar}|$ records to be sent to the expert together with ar .

As another variation, the methodology may handle cases where the throughput capacity $HITL$ of rules that can be checked by the humans involved in the process is unknown. In this case, the amount cvf of association rules may be given as input, together with the fraction of rules to be tested l . Randomly select a set $G \subseteq AR$ of cardinality cvf to be sent to the crowd of experts. In this variation, it is not guaranteed that all the association rules of G will be verified. Rather, a best-effort approach may be carried on by the human component.

A further variation may require the additional input parameters $support_verification_threshold \in [0, 1]$, $confidence_verification_threshold \in [0, 1]$ and $record_verification_threshold \in [0, 1]$. During the selection of the rules (and, possibly, records) to be sent to the experts, select only those rules ar (records r) for which:

$$\begin{aligned} S(AV) &< support_verification_threshold, \\ C(AV) &< confidence_verification_threshold, \\ t_r &< record_verification_threshold, \end{aligned}$$

where AV is a set containing both the ground atoms in the body and the head of ar . Thus, only the rules and records that do not fulfil the above criteria of trust will be sent to experts, whereas the remaining rules will be automatically trusted and enforced.

Examples of Alternative of Expert Selection (142)

For evaluating the association rules resulting from the Rule Cleaning (130) based on domain expert knowledge do the following. For grounded rules of G , instead of selecting experts (from the Human In The Loop team) based on their knowledge and/or previous performance, experts are selected randomly.

Examples of Alternative of Results Evaluation (143)

As a variation, the accuracy metric can be replaced with other known metrics to evaluate the rule set G .

Other variations include the specification of a *rule evaluation strategy* to handle user-specified policies. This means that not only rule sets but also single rules are evaluated and/or rated by experts. In case a single rule is evaluated by two or more experts, the rule evaluation strategy aggregates over these multiple evaluations and thus retains a single aggregate evaluation per rule.

In some examples, in the *rule evaluation strategy*, the rule set evaluation is actually mainly based on single-rule evaluations, while in other examples, the rule-set evaluation is replaced by the evaluation of single rules. Further examples of a possible *rule evaluation strategy* use single-rule evaluation as a supplementary feature, in addition to rule set evaluation.

When single rules are evaluated by experts, it makes sense to retain rules that are positively evaluated for further rounds of the computation. In what follows, an example of a *rule evaluation strategy* that retains positively evaluated rules over several iterations of the procedure is described. For every evaluated association rule $ar \in G$, compute the accuracy of ar as the following:

$$accur(ar) = \frac{\text{number of positive expert validations of } ar}{\text{number of all expert validations of } ar} \quad (5.9)$$

Depending on the evaluation of the rule set, the procedure then distinguishes two cases:

1. $accur(G) < p$, which means that the rule set evaluation is not successful, and the accuracy for a given rule ar is greater than a user-specified threshold, store ar in an ordered set RAR , with a descending order based on $accur(ar)$. In this case, the subsequent iterations of the $ARM(D, I)$ procedure start with $AR := AR \cup RAR$. In the case that at any point in the procedure $|AR \cup RAR| >$

nr_{max} , then association rules with lower accuracy values in RAR are dropped until $RAR = \emptyset$ or $|AR \cup RAR| = nr_{max}$.

2. $accur(G) \geq p$, which means that the rule set evaluation is successful, then all the association rules $ar \in AR$ for which $accur(ar)$ is below a user-specified threshold, are removed from G .

In other *rule evaluation strategy* examples, the set RAR might be exempted from the requirement to meet the upper bound on the number of association rules, i.e. only the set of association rule AR generated during the current iteration of the procedure will be accounted for the threshold nr_{max} .

Rule Compression

Here, we introduce a methodology for creating a rule-forest structure that embeds all the association rules. The creation of the rule-forest structure is part of the Rule Evaluation **140** process. This factorizes common elements between separate association rules, producing a space-efficient structure. The produced rule-forest can be used by domain experts to get a broad yet compact overview of the association rules set and the discriminant between multiple association rules.

The computation performed by the system iterates over all the mined rules $ar \in AR$ to create a decision rule-forest structure $forest(AR)$. The rules are divided into sets where there exists at least one ground atom appearing in the body of every rule in the set. For example, considering the set of rules **800** in Figure 5.2, the rules can be divided into two sets: one containing the rules **801** and **802** and the other containing the rules **803** and **804**.

A decision rule-tree is created for each set sr where each node of the tree is composed of a ground atom appearing in the bodies of the association rules in sr , and each leaf is composed of a ground atom appearing in the heads of the association rules in sr . Figure 5.2 shows an example of the creation of a rule-forest structure (**830**) starting from a given set of rules (**800**). The rule-forest is composed of two distinct rule-trees (**810** and **820**).

In order to create the rule-tree **810**, the rules shown in the elements **801** and **802** are utilised. First, the common body atom $\#employees(C_1)$ is utilised to create the node **811**. Then, since also the second ground atom in the body $income(C_2)$ is shared, this is used to create the node **812** connected via a direct arc from **811**. Given that there is no other common ground atom between the considered rules,

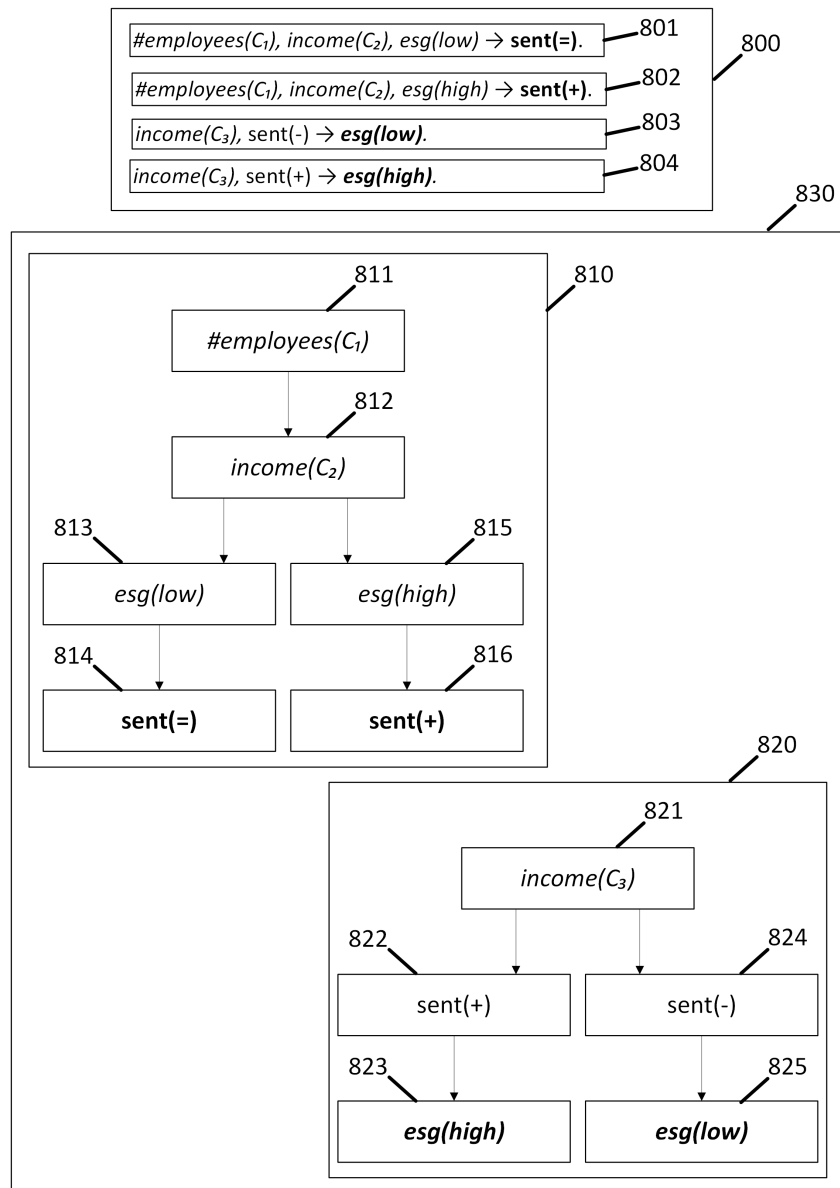


Figure 5.2: This Figure depicts an example of Rule Compression, including two rule-trees, which together constitute a rule-forest according to an example of the instant disclosure. The rule-trees are generated from a set of association rules.

the rest of the ground atoms in the body are mapped to elements on separate directed paths originating from **812**. Therefore, two different elements, **813** and **815** containing, respectively, the ground atoms *esg(low)* and *esg(high)* are created. Finally, the ground atoms in the heads of, respectively, **801** and **802** are used to create the leaves **814**, with a direct path from **813**, and **816**, with a direct path from **815**, containing the ground atoms *sent(=)* and *sent(+)*. Note that both the head of the rules and the text in the leaves are in bold text to highlight that they refer to the same concept, i.e. the consequent of a rule. Likewise to the creation of rule-tree **810**, the rule-tree **820** is constructed by using the rules in **803** and **804**. The node **821** corresponds to the first body atom *income(C₃)* of both **803** and **804**. The nodes **822** and **824** represent the second body atom *sent(+)* and *sent(-)* of, respectively, **804** and **803**. Since **822** and **824** are not equal, two different directed paths are started from **821** connecting to them. Finally, **823** and **825** represent, respectively, the body heads of **804** (*esg(high)*) and **803** (*esg(low)*). **823** is connected to a directed arc originating in **822**, while **825** is connected to a directed arc originating in **824**. Together, rule-trees **810** and **820**, constitute the rule-forest **830**.

Each directed path from the root of a rule-tree to a leaf constitutes an association rule. In the same rule-tree, different root-leaf paths necessarily contain different nodes corresponding to different ground atoms in the relative association rules. These ground atoms are the discriminants that the association rules use to make different predictions.

Several algorithms and optimizations for constructing *forest(AR)* may be used in different iterations and/or in different variations of the procedure, leading to the construction of different rule-forests even in the presence of the same set of association rules.

Rule-Forest Integrity Check

The procedure, within the context of the Rule Evaluation **140** process, sends the decision rule-forest structure *forest(AR)* computed during the Rule Compression to an appropriate subset of the available experts in the Human In The Loop team for an integrity check of the overall program. A consensus algorithm is run to find an agreement among the experts involved. If a positive consensus is not reached, which indicates that possibly *forest(AR)* badly represents the application domain, the computation is restarted from Uncertain Record Elimination (**115**), where a

Parameter Modification over the input parameters is performed as follows:

$$\begin{aligned}
 m &:= m + (m * ss_m) \\
 t &:= t + (t * ss_t) \\
 S &:= S + (S * ss_S) \\
 C &:= C + (C * ss_C) \\
 nr_{min} &:= nr_{min} - (nr_{min} * ss_{nr_{min}}) \\
 K &:= K + (K * ss_K) \\
 KIR &:= KIR + (C * ss_{KIR}) \\
 p &:= p + (p * ss_p)
 \end{aligned}$$

In case a positive consensus is reached, the process continues.

A large number of alternatives are possible and may or may not be appropriate in various application contexts. For example, in some contexts, it may be useful to increment or decrement the parameter l , too.

5.1.10 Rule Application (150)

The process "Rule Application", mentioned in Figures 5.1, A.1, and A.6, is responsible to make use of the mined rules. The set of mined association rules AR is used to improve the quality of the original database D_0 (**101**). For each ground atom $a(v)$ inferred by the application of an association rule ar (i.e. $a(v) = head(ar)$) to a record $r \in D_0$, perform one of the following:

1. If the record r used for the association rule ar contains a conflicting value $val(a, r) \neq a(v)$ (**151** and **152**), then Outlier Detection and Correction (**153**) is performed to correct r .
2. If the record r used for the association rule ar contains a concordant value $val(a, r) = a(v)$ (**151** and **152**) and $val(t_a, r) \neq 1$, then Truth Reinforcement (**154**) may be performed to reinforce the truth value of a in r . If $val(t_a, r) = 1$, nothing is performed to preserve original ground knowledge.
3. Otherwise, if r contains a null value for $val(a, r)$ (**151**), then perform Data Completion (**154**) to complete r .

The above-mentioned processes are now described in order with more detail.

Outlier Detection and Correction (153)

For each association rule $ar \in AR$ of the form shown in Equation 5.1, find all the records $r \in D_0$ such that $body(ar) \in r$ and $head(ar) = a(v)$ and such that $val(a, r) \neq a(v)$ and $val(a, r) \neq NULL$ (**151** and **152**). The procedure performs the following updates: $val(a, r) := a(v)$ and $val(t_a, r) := C(ar)$.

Truth Reinforcement (154)

For each association rule $ar \in AR$ of the form shown in Equation 5.1, find all the records $r \in D_0$ such that $body(ar) \in r$ and $head(ar) = a(v)$ and such that $val(a, r) = a(v)$ and $val(t_a, r) \neq 1$ (**151** and **152**). The procedure performs the following update:

$$val(t_a, r) := \max \left\{ \frac{tr_{old} * val(t_a, r) + tr_{new} * C(ar)}{tr_{old} + tr_{new}}, val(t_a, r) \right\}, \quad (5.10)$$

where $C(ar)$ is the fraction expressing confidence of ar , tr_{old} and tr_{new} are the weights indicated in I (**102**) to balance the update.

Data Completion (155)

For each association rule $ar \in AR$ of the form shown in Equation 5.1, find all the records $r \in D_0$ such that $body(ar) \in r$ and $head(ar) = a(v)$ and such that $val(a, r)$ is not defined or contains a null value (**151**). Then, the procedure performs the following update: $val(a, r) := a(v)$.

5.1.11 Examples of Variations of Rule Application (150)

Here, we present a list of possible variations that can be applied to the process of Rule Application (**150**).

Examples of Alternative of Outlier Detection and Correction (153)

For each association rule $ar \in AR$ of the form shown in Equation 5.1, find all the records $r \in D_0$ such that $body(ar) \in r$ and $head(ar) = a(v)$ and such that $val(a, r) \neq a(v)$ and $val(a, r) \neq NULL$. The procedure, instead of overwriting the original attribute value, adds a new column (if not already present) $a_corrected$ to D_0 (**101**) and performs the following update: $val(a_corrected, r) := a(v)$.

In other examples, if it is not already present, a column t_a is added to D_0 . The t_a -value of a record r can be computed as a (possibly weighted) combination of other truth values and parameters.

Examples of Alternative of Truth Reinforcement (154)

For each association rule $ar \in AR$ of the form shown in Equation 5.1, find all the records $r \in D_0$ such that $body(ar) \in r$ and $head(ar) = a(v)$ and such that $val(a, r) = a(v)$ and $val(t_a, r) \neq 1$. Instead of performing the update as indicated herein, using the original truth value $val(t_a, r)$ and the confidence of the association rule ar , the method may consider a different combination of metrics for the update, such as, for example, a weighted combination of confidence, support, Kulc, IR, and $val(t_a, r)$.

Examples of Alternative of Data Completion (155)

For each association rule $ar \in AR$ of the form shown in Equation 5.1, find all the records $r \in D_0$ such that $body(ar) \in r$ and $head(ar) = a(v)$ and such that $val(a, r)$ is not defined or contains a null value. The procedure, instead of overwriting the original attribute value, adds a new column (if not already present) $a_inferred$ to D_0 (101) and performs the following update: $val(a_inferred, r) := a(v)$.

Explainability feature

When performing one of the subprocesses of Rule Application (150), the records in the database D_0 (101) may be further enhanced with an explanation ex relative to the inferred data. This can be performed by introducing a new column with a fresh attribute name $expl_a$ in D_0 , where a is the attribute which $expl_a$ provides an explanation of. Given the completion $val(a, r) = a(v)$ made by rule ar over the record r , write an explanation for the completion in the column $expl_a$ using the values of $body(ar)$ together with the quality metrics computed during $ARM(D, I)$, such as support, confidence, Kulc, IR and accuracy. For example, given an association rule $ar = \#employees(C_i), income(C_j) \rightarrow esg(low)$, with associated metrics $S(ar) = 0.2, C(ar) = 0.8, Kulc = 0.95, IR = 0.5, accur = 0.8$, where $\#employees$ and $income$ use generic indexes to keep the example more abstract, the content of $val(expl_a, r)$ for a record r may, for example, look as follows:

"The value of esg for the record $\mathbf{r}$ has been predicted as **low**, given that (i) the value of $\#employees$ is equal to C_i , (ii) the value of $income$ is equal to C_j , (iii) the rule $\mathbf{ar}$ is $\#employees(C_i), income(C_j) \rightarrow esg(low)$ and (iv) the metrics associated to $\mathbf{ar}$ are: $S(ar) = 0.2, C(ar) = 0.9, Kulc(ar) = 0.95, IR(ar) = 0.5, accur = 0.8$."

Introduction of an additional fuzzy feature

Another variation extends the above described enhanced explanation by a truth value $val(t_a, r)$ for the predicted value $val(a, r)$, where $a \in head(ar)$. In this variation, the input I (**102**) contains, as an additional parameter, a weight $w_m \in [0, 1]$ associated to each metric m . The value for $val(t_a, r)$ is based on the support S , confidence C , Kulczynski measure $Kulc$, imbalance ratio IR and accur p , and is computed as follows:

$$val(t_a, r) = \frac{(S * w_S) + (C * w_C) + (Kulc * w_{Kulc}) + (IR * w_{IR}) + (p * w_p)}{w_S + w_C + w_{Kulc} + w_{IR} + w_p}. \quad (5.11)$$

In other variations, other metrics and methods of other methods of aggregating those metrics might be used, for example, a weighted combination of t-norms may be used for the aggregation.

Introduction of an additional usability feature

Let D' be a database that is different from D_0 (**101**) and unrelated to $ARM(D, I)$, yet still having the same schema as A or a schema compatible with A . The application of the mined association rules AR during Rule Application (**150**) can be performed over D' to reuse the knowledge mined from D_0 .

Introduction of an additional optimization

Instead of applying the mined association rules one at a time, the rule-trees in $forest(AR)$ may be used for faster computation, reducing the overall amount of comparisons between rule bodies and records.

5.1.12 Introduction of a variation for multiple procedures: Parameter Modification

As another example, upon violation of the quality thresholds indicated in the input such as, for example, nr_{min} or p , may perform the following. Instead of relaxing all the input parameters at once before restarting the computation, undertake a more flexible approach than the one described in Mining Evaluation (**122**), Results Evaluation (**143**) and Rule-Forest Integrity Check, as explained below.

The required Parameter Modification may be performed in a Gradient Descent fashion by first probing and then relaxing one input parameter at a time. This input parameter is chosen amongst the ones responsible for the failure to reach

the threshold requirements, i.e. those parameters already mentioned in the Modification. The relaxation that shows the best improvements during the test is then applied. This is repeated until all the thresholds are satisfied.

Extensions, Refinements, Variations and Alternatives

The above-disclosed process $ARM(D, I)$ and some alternative processes have been presented as an example. Some of the processes introduced are purposely based on arbitrary algorithmic, input and data structures used to present a practical yet general enough picture of the overall methodology.

Some of the processes shown in the following are not strictly necessary for the termination of the procedure and the computation of association rules. For example, excluding Symmetric Functional Dependencies Removal (**112**) and/or Uninteresting Attribute Removal (**113**) may still allow the termination of the process, possibly with some inefficiencies. Likewise for Attribute Discretization (**114**), with the difference that the inefficiencies introduced by the removal of Attribute Discretization would be so strong to prevent most of the computations from ending in a reasonable time. The subprocesses Uncertain Record Elimination (**115**) and Record t-Fuzzification (**116**) are needed only in the eventuality that the database contains fuzzy data. However, the fuzziness of the data could be ignored, and an approximated computation could be performed, making use of Truth Degrees Removal (**117**) to remove the undesired truth values and sacrificing the accuracy of the results produced. The subprocess of Mining Evaluation (**122**) could be optionally removed, likewise with the Rule Cleaning (**130**) and the whole process of Rule Evaluation (**140**). This would drastically reduce the quality of the results obtained by the process Rule Application (**150**) in favour of a lighter computation.

5.2 An Industry-Led Use Case

In this section, we report on an industry application of the method $ARM(D, I)$ introduced in Section 5.1. This has been part of a larger project whose aim was the verification and completion of Knowledge Graphs with company data [75, 81].

Companies' data constitute information significantly valuable for many business applications. Meltwater [119] provides users with access to a large Knowledge Graph, Owler [169, 130], which contains useful information about companies amongst which companies' revenue, number of employees, sector and competitor pairs. Owler's data are crowd-sourced from over a million contributors.

Crowd-sourced data are inherently incomplete and inaccurate. These data are usually obtained by creating tasks for each data completion or verification required and sending the same task to a plethora of experts. Then, once all the experts have manually inserted the required data, a consensus algorithm decides on the values to be inserted in the KG. Unfortunately, with hundreds of millions of companies around the world (as reported by *Statista.com* [155]), it would be unfeasible to pay multiple experts for each task of completion or revision for each data point of every company. Moreover, finding a consensus among experts is not easy. This is even more true when these data points are not easy to discover even for humans and may change constantly over time. For instance, an expert could say that a company works in the health sector because it sells band-aids, even if its main products are perfumes.

Therefore, automatically inferring and verifying company data represents a crucial task.

Automatic extraction of statistically relevant logical rules requires huge input data. Thanks to the industrial collaboration with Meltwater, we could apply the methodologies of $ARM(D, I)$ using the Owler KG, maintained by Meltwater, which contains data for around 18 million companies.

The experiments presented in this section were conducted using Owler KG, a proprietary dataset provided by Meltwater, under a non-disclosure agreement. To protect the confidentiality and privacy of Meltwater and its customers, the complete schema and details of the attributes are not reported here.

We here present some of the main characteristics of Owler KG which are relevant for the purposes of the present section:

- The nodes represent companies, and the edges represent relations between them, such as competitors, subsidiaries, or industry classifications.
- The nodes have various attributes, such as core firmographics, CEO, HQ location, revenue, and employee count. These attributes are partially validated by humans and expected to be reasonably complete, but their completeness varies across the graph. For example, all nodes have logo, name, and URL, but only about 50% have headquarters, about 25% have revenue and/or employee estimates and only about 10% have more than one competitor. Some of the attribute values are paired with a fuzzy value in the domain $[0, 1]$, indicating the confidence in the correctness of the attribute value.
- The majority of the nodes and edges have little human verification. This means that there may be errors, inconsistencies, or missing information in the graph.
- A sub-portion of the graph, containing the most important companies, is continuously verified and maintained.

We use the methodologies explained in $ARM(D, I)$ to discover frequent patterns and rules in the graph and employ them to predict and correct data points. We evaluate the predictions' performance and usefulness via manual inspections performed by Meltwater's experts on the Owler KG.

Using the data preparation process, we reduced the input size by removing symmetrical functional dependencies such as

$$\{Company_ID \rightarrow Company_name, Company_ID \leftarrow Company_name\}$$

and by removing attribute values and records with a low truth value. After this, the truth degree of the remaining records has been removed. Moreover, we discretized the domains of several numerical attributes such as *Revenue* and *#_employees* into a finite number of labels indicating ranges of these attributes. We repeated this process with ranges of size 3, 5 and 10 to generate both general and more specific rules. We also removed uninteresting attributes such as *Company_ID* as these won't be helpful for the purpose of finding patterns.

After the data preparation, a standard Apriori algorithm is applied, using the workbench Weka [67]. From this, we mined almost 1000 association rules, which, after the rule cleaning and evaluation processes, have been filtered down to

81. The majority of these are used to infer ranges of number of employees and company revenues. We here report two of these rules:

```

revenue(Company_ID, "1B <= REV. < 10B"),
    exchange(Company_ID, "NYSE")
→ employees(Company_ID, "100 < EMP.").
employees(Company_ID, "1000 < EMP. <= 10000"),
    country(Company_ID, "USA")
→ revenue(Company_ID, "50M <= REV.").

```

More association rules are available in the Appendix to this document.

Using the mined rules, we produced ~18000 "number of employees" range predictions and ~1000 "company revenue" range predictions to complete Owler's data. These are of one of the two following forms:

```

1 {
2   "company": ID,
3   "predicted_revenue": PRED,
4   "confidence": CONF,
5   "explanation": EXPL
6 },
7 {
8   "company": ID,
9   "predicted_employees": PRED,
10  "confidence": CONF,
11  "explanation": EXPL
12 }

```

where *ID* is an identifier of the company, *PRED* is a label amongst the discretized ones such as "100< EMP." or "50 M<= REV.", *CONF* $\in [0, 1]$ reflects the confidence in the prediction and *EXPL* is a string providing an explanation of the predicted value for future reviewers. An example of *EXPL* is "The value of 'REVENUE' is predicted as '50 M<= REV.' by the rule 'revenue00038' with confidence of 99%, given that 'COUNTRY'='USA', 'EMPLOYEE'='1000< EMP. <=10000', valid for this record, is/are found to be true #3490 times in the original dataset".

Moreover, we also obtained ~50000 "number of employees" range smells and ~16000 "company revenue" range data smells to repair Owler's data. These are of one of the two following forms:

```

1 {
2   "company": ID,
3   "expected_revenue": EXP,
4   "actual_revenue": ACT,
5   "confidence": CONF,
6   "explanation": EXPL
7 },
8 {
9   "company": ID,
10  "expected_employees": EXP,
11  "actual_employees": ACT,
12  "confidence": CONF,
13  "explanation": EXPL
14 }

```

where *ID* is an identifier of the company, *EXP* is a label amongst the discretized ones such as "100< EMP." or "50 M<= REV.", $ACT \in \mathbb{N}$, $CONF \in [0, 1]$ reflects the confidence in the prediction and *EXPL* is a string providing an explanation of the predicted value for future reviewers such as "The value of 'EMPLOYEE' is predicted as '100< EMP.' by the rule 'employee00352' with a confidence of 97%, given that 'REVENUE'='50 M<= REV.', 'FOUNDED'='20<AGE', valid for this record, is/are found to be true #12553 times in the original dataset".

Note that for both data smells and data predictions, *EXPL* is automatically generated by expanding the information of the association rules in a format accessible to non-experts.

A sample of 200 data smells and 200 data predictions has been evaluated by Meltwater's experts. These found an accuracy of 69% for the predictions, 96% for the data smells, and an overall 82% accuracy across all the predictions and data smells. These results are reported in Figures 5.3 and 5.4.

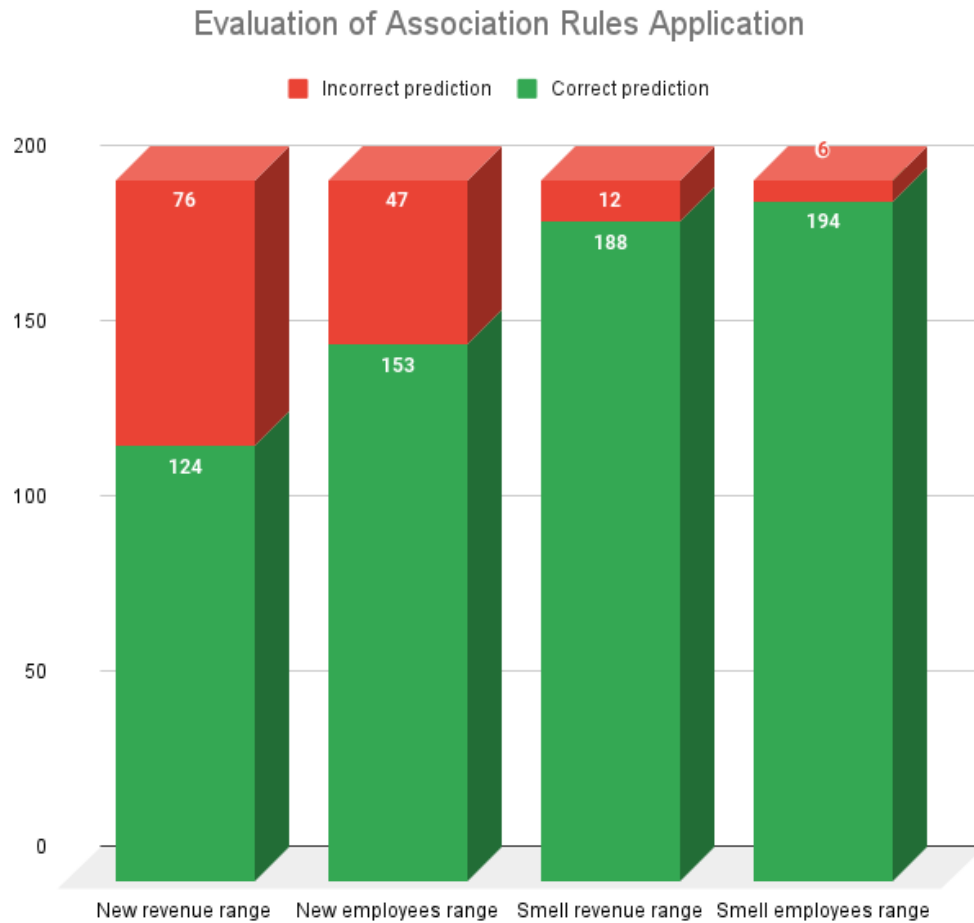


Figure 5.3: Results of the evaluation of the application of mined Association Rules to the Owler KG.

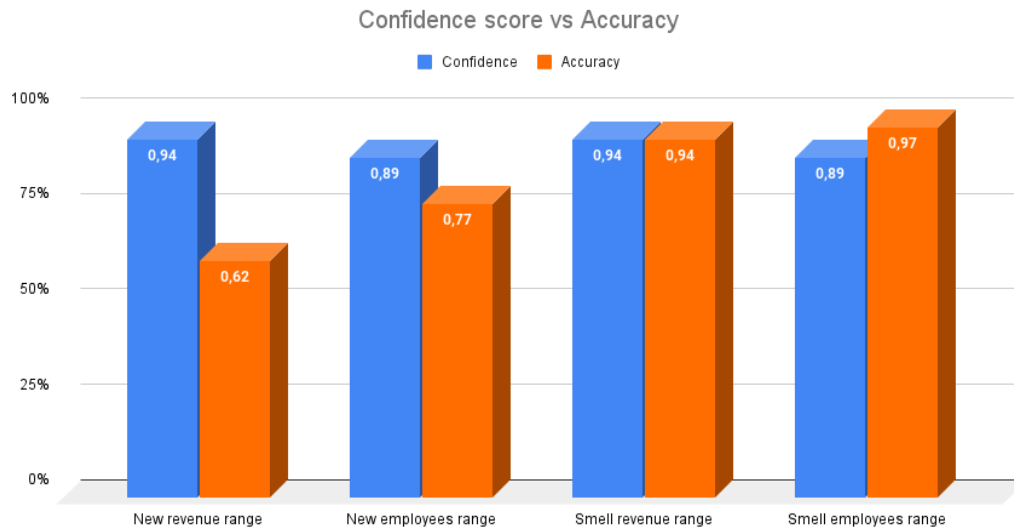


Figure 5.4: Comparison of the aggregated confidence of the predictions/smells provided, with accuracy observed by Meltwater’s experts.

5.3 Summary

This chapter presents a novel, patented method for automatically mining high-quality association rules named $ARM(D, I)$. It addresses the challenge of reducing the workload of creating domain expert rules. $ARM(D, I)$ offers a comprehensive framework for mining rules from various data types, including crisp and fuzzy data. It allows for human validation when available and incorporates automatic parameter tuning to ensure an efficient rule discovery.

The system operates through five key processes: data preparation, rule mining, rule cleaning, rule evaluation, and rule application. Each step is designed to produce high-quality rules with customizable options through parameter tuning for specific use cases.

The data preparation process prepares the input dataset for a more effective and accurate association rules mining. This eliminates irrelevant information and low-quality fuzzy data and discretizes continuous data for optimal mining. The core $ARM(D, I)$ mining process discovers rules that meet user-specified criteria, such as support, confidence, and rule quantity. Following mining, a rule-cleaning step enforces additional constraints for ruling out statistically irrelevant rules. The

rule evaluation process integrates human expertise through functionalities like workload estimation, expert selection for review tasks, and advanced visualization tools. Finally, the system incorporates methods to automatically apply the mined rules to repair and increase the coverage of the original dataset.

The chapter further showcases the system's application in an industry setting, using a crowd-sourced dataset of companies from Meltwater. We demonstrate how $ARM(D, I)$ uncovers valuable patterns that improve data quality and facilitate data maintenance. Collaboration with Meltwater's specialists confirmed the suitability of these predictions and data anomalies for automated integration into production systems.

CHAPTER 6

Conclusions

This thesis addressed the challenges and opportunities present within rule-based reasoning in the context of large and many-valued datasets. It demonstrates the power of new Datalog-based formalisms to model many-valued settings, how to achieve computational efficiency through program rewritings, and the potential for automated rule mining in revealing underlying patterns and regularities within crisp and/or uncertain data.

In Chapter 3, we presented the novel semantics of MV-Datalog and MV-Datalog[±] as the fuzzy extensions of, respectively, Datalog and Datalog[±]. After characterizing the notion of minimal fuzzy models for Datalog as the models in which every fact takes the least truth degree, we proved that unique minimal fuzzy models always exist and can be found by solving a linear programming problem. This is also true for K -fuzzy models, in which each rule of an MV-Datalog program is K -satisfied rather than completely satisfied. We prove that fact-entailment in MV-Datalog retains the same complexity as fact-entailment in Datalog. To allow reasoning in the context of Datalog[±], we introduced the notion of preferred models by leveraging a new definition of strong existential quantification. We defined a practice-oriented approach to find preferred models in practical settings. We proved the effectiveness of MV-Datalog by addressing a link-prediction problem with many-valued big data.

In Chapter 4, we focused on optimising query answering under the Piece-Wise Linear $\cup$ Warded Datalog[±] fragment in both crisp and many-valued settings. We proved that PWLW programs can be rewritten in a linear normal form, which retains CQ-equivalence in both the crisp and many-valued settings while improving CQ-answering performances. Moreover, the linear normal form paves the way for a more straightforward design and implementation of more efficient parallel operators. We demonstrated the advantages of the linear normal form by experimenting with synthetic scenarios.

In Chapter 5, we presented a patented method on association rules which extends the reach of logical reasoning systems, enabling the automation of the mining, evaluation and application of meaningful relationships within complex crisp and/or fuzzy datasets. We proved the effectiveness of such methodology in an industrial scenario to predict and correct a knowledge graph.

Limitations and Future Directions This thesis identified several areas for further exploration that build upon the presented work:

- **Generalization of MV-Datalog and MV-Datalog[±] Semantics:** The current semantics of MV-Datalog and MV-Datalog[±] are restricted to using the Łukasiewicz t-norm. We worked towards allowing the use of arbitrary t-norms and stratified negation, which are out of the scope of this thesis [102]. The results achieved so far focus on programs that allow for a finite chase. However, relevant fragments of Datalog[±] exist, such as (weakly) guarded Datalog[±] or warded Datalog[±], where entailment remains decidable despite a potentially infinite chase. Addressing entailment for fuzzy Datalog and Datalog[±] programs in the presence of infinite chase is an interesting future direction.
- **Advanced Linearization Techniques:** This thesis demonstrated the effectiveness of linearization techniques in both crisp and fuzzy settings for conjunctive query (CQ) answering. Developing specialized optimizations, such as leveraging parallel operators, shows potential for further performance enhancements.
- **Improving Efficiency of $ARM(D, I)$:** The $ARM(D, I)$ system relies on traditional association rule mining algorithms like Apriori and FP-Growth, leading to potentially high computational costs. Integrating or developing more efficient association rule mining algorithms would significantly improve overall

performance and enable better scalability. In addition, when performing automatic parameter tuning, implementing result-caching mechanisms could prevent redundant association rule recomputations, leading to improved efficiency.

Concluding Remarks This thesis contributes significantly to bridging the gap between symbolic and subsymbolic reasoning, towards an effort to integrate these two. By expanding the capabilities of logical reasoners to handle many-valued settings in an efficient and automated manner, the potential applications become broader, the reasoning processes more reflective of real-world complexities, and the insights generated more actionable.

This thesis also opens up new avenues for research and innovation in the field of rule-based reasoning, as well as in the broader field of artificial intelligence. It demonstrates the power and versatility of logic as a tool for modelling, reasoning, and learning from data, and the potential for synergy and complementarity between symbolic and subsymbolic approaches.

Bibliography

- [1] OpenCyc. <https://www.cyc.com/>.
- [2] Google Knowledge Graph. <https://developers.google.com/knowledge-graph>, 2012.
- [3] GUIDELINE (EU) 2011/14 OF THE ECB. https://www.ecb.europa.eu/ecb/legal/pdf/L_33120111214en000100951.pdf, 2014.
- [4] Serge Abiteboul, Richard Hull, and Victor Vianu. *Foundations of Databases*. Addison-Wesley, 1995.
- [5] Attila Achs, Agnes Kiss. Fuzzy extension of datalog. *Acta Cybernetica*, 12(2):153–166, 1995.
- [6] Foto N Afrati, Manolis Gergatsoulis, and Francesca Toni. Linearisability on datalog programs. *Theor. Comput. Sci.*, 308(1-3):199–226, 2003.
- [7] Rakesh Agrawal, Tomasz Imielinski, and Arun N Swami. Mining Association Rules between Sets of Items in Large Databases. In *Proceedings of ACM SIGMOD*, pages 207–216. ACM Press, 1993.
- [8] Rakesh Agrawal and Ramakrishnan Srikant. Fast Algorithms for Mining Association Rules in Large Databases. In *Proceedings of VLDB*, pages 487–499. Morgan Kaufmann, 1994.

- [9] Alfred V Aho, Catriel Beeri, and Jeffrey D Ullman. The Theory of Joins in Relational Databases. *ACM Trans. Database Syst.*, 4(3):297–314, 1979.
- [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 *Proceedings of EuroSys*, pages 223–236. ACM, 2010.
- [11] Stig K Andersen. Judea Pearl, Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference. *Artif. Intell.*, 48(1):117–124, 1991.
- [12] 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 *SIGMOD*, pages 1371–1382, 2015.
- [13] Marcelo Arenas, Pablo Barceló, Leonid Libkin, Wim Martens, and Andreas Pieris. *Principles of Databases*. Open source at <https://github.com/pdm-book/community>, 2021.
- [14] Marcelo Arenas, Pablo Barceló, Leonid Libkin, Wim Martens, and Andreas Pieris. *Database Theory*. Open source at <https://github.com/pdm-book/community>, 2022.
- [15] Paolo Atzeni, Teodoro Baldazzi, Luigi Bellomarini, and Emanuel Sallinger. iWarded: A Versatile Generator to Benchmark Warded Datalog+/- Reasoning. In *Proceedings of RuleML+RR*, volume 13752 of *Lecture Notes in Computer Science*, pages 113–129. Springer, 2022.
- [16] Paolo Atzeni, Luigi Bellomarini, Michela Iezzi, Emanuel Sallinger, and Adriano Vlad. Weaving Enterprise Knowledge Graphs: The Case of Company Ownership Graphs. In *EDBT*, pages 555–566, 2020.
- [17] Stephen H Bach, Matthias Broecheler, Bert Huang, and Lise Getoor. Hinge-Loss Markov Random Fields and Probabilistic Soft Logic. *J. Mach. Learn. Res.*, 18:109:1–109:67, 2017.
- [18] Jean-François Baget, Michel Leclère, Marie-Laure Mugnier, Swan Rocher, and Clément Sipieter. Graal: A Toolkit for Query Answering with Existential Rules. In *RuleML*, pages 328–344, 2015.

- [19] Jean-François Baget, Marie-Laure Mugnier, Sebastian Rudolph, and Michaël Thomazo. Walking the Complexity Lines for Generalized Guarded Existential Rules. In *Proceedings of IJCAI*, 2011.
- [20] Collin F Baker, Charles J Fillmore, and John B Lowe. The Berkeley FrameNet Project. In *Proceedings of COLING-ACL*, pages 86–90. Morgan Kaufmann Publishers / ACL, 1998.
- [21] Teodoro Baldazzi, Luigi Bellomarini, Emanuel Sallinger, and Paolo Atzeni. Eliminating Harmful Joins in Warded Datalog+/- . In *Proceedings of RuleML+RR*, pages 267–275. Springer, 2021.
- [22] F Barca and M Becht. The Control of Corporate Europe. Oxford University Press, European Corporate Governance Network. 2001.
- [23] Pablo Barceló and Reinhard Pichler. *Datalog in Academia and Industry: Second International Workshop, Datalog 2.0, Vienna, Austria, September 11-13, Proceedings*, volume 7494. Springer, 2012.
- [24] Catriel Beeri and Moshe Y Vardi. A Proof Procedure for Data Dependencies. *J. ACM*, 31(4):718–741, 1984.
- [25] Luigi Bellomarini, Davide Benedetto, Georg Gottlob, and Emanuel Sallinger. Vatalog: A modern architecture for automated reasoning with large knowledge graphs. *Inf. Sys.*, 2020.
- [26] Luigi Bellomarini, Georg Gottlob, Andreas Pieris, and Emanuel Sallinger. Swift Logic for Big Data and Knowledge Graphs - Overview of Requirements, Language, and System. In *Proceedings of SOFSEM*, volume 10706 of *Lecture Notes in Computer Science*, pages 3–16. Springer, 2018.
- [27] Luigi Bellomarini, Georg Gottlob, Andreas Pieris, and Emanuel Sallinger. The Vatalog System: Swift Logic for Big Data and Enterprise Knowledge Graphs. In *Proceedings of AMW*, volume 2100 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2018.
- [28] Luigi Bellomarini, Georg Gottlob, Andreas Pieris, and Emanuel Sallinger. Vatalog: A Language and System for Knowledge Graphs. In *Proceedings of RuleML+RR*, volume 11092 of *Lecture Notes in Computer Science*, pages 3–8. Springer, 2018.

- [29] Luigi Bellomarini, Eleonora Laurenza, Emanuel Sallinger, and Evgeny Sherkhonov. Reasoning Under Uncertainty in Knowledge Graphs. In *Proceedings of RuleML+RR*, volume 12173 of *Lecture Notes in Computer Science*, pages 131–139. Springer, 2020.
- [30] Luigi Bellomarini, Emanuel Sallinger, and Georg Gottlob. The Vatalog System: Datalog-based Reasoning for Knowledge Graphs. *Proceedings of VLDB*, 11(9):975–987, 2018.
- [31] Gerald Berger, Georg Gottlob, Andreas Pieris, and Emanuel Sallinger. The Space-Efficient Core of Vatalog. In *Proceedings of ACM PODS*, pages 270–284. ACM, 2019.
- [32] Marta Bernardini, Matthias Lanzinger, Rosario Laurendi, and Stefano Sferazza. Family Link Detection in Uncertain Settings with MV-Datalog+-. In *EDBT/ICDT Workshops*, 2022.
- [33] David M Blei, Alp Kucukelbir, and Jon D McAuliffe. Variational Inference: A Review for Statisticians. *CoRR*, abs/1601.0, 2016.
- [34] Fernando Bobillo and Umberto Straccia. The fuzzy ontology reasoner fuzzyDL. *Knowl. Based Syst.*, 95:12–34, 2016.
- [35] Stefan Borgwardt, Marco Cerami, and Rafael Peñaloza. Łukasiewicz Fuzzy EL is Undecidable. In *Proceedings of DL*, volume 1879 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2017.
- [36] Stefan Borgwardt and Rafael Peñaloza. Fuzzy Description Logics - A Survey. In *Proceedings of SUM*, volume 10564 of *Lecture Notes in Computer Science*, pages 31–45. Springer, 2017.
- [37] Andrea Calì, Georg Gottlob, and Michael Kifer. Taming the infinite chase: Query answering under expressive relational constraints. *Proceedings of JAIR*, 48:115–174, 2013.
- [38] Andrea Calì, Georg Gottlob, and Thomas Lukasiewicz. A general Datalog-based framework for tractable query answering over ontologies. *Proceedings of JoWS*, 14:57–83, 2012.
- [39] Andrea Calì, Georg Gottlob, and Andreas Pieris. Towards more expressive ontology languages: The query answering problem. *Artificial Intelligence*, 193:87–128, 2012.

- [40] Andrea Calì, Georg Gottlob, and Andreas Pieris. Towards more expressive ontology languages: The query answering problem. *Proceedings of JAIR*, 193:87–128, 2012.
- [41] David Carral, Irina Dragoste, Larry González, Criel J H Jacobs, Markus Krötzsch, and Jacopo Urbani. VLog: A Rule Engine for Knowledge Graphs. In *Proceedings of ISWC*, volume 11779 of *Lecture Notes in Computer Science*, pages 19–35. Springer, 2019.
- [42] Roger Cavallo and Michael Pittarelli. The Theory of Probabilistic Databases. In *Proceedings of VLDB*, pages 71–81. Morgan Kaufmann, 1987.
- [43] Stefano Ceri, Georg Gottlob, and Letizia Tanca. What you always wanted to know about Datalog (and never dared to ask). *TKDE*, 1(1):146–166, 1989.
- [44] Stefano Ceri, Georg Gottlob, and Letizia Tanca. *Logic Programming and Databases*. Springer, 1990.
- [45] Ariane Chapelle and Ariane Szafarz. Controlling firms through the majority voting rule. *Physica A: Statistical Mechanics and its Applications*, 355(2-4):509–529, 2005.
- [46] Brian Chin, Daniel von Dincklage, Vuk Ercegovic, Peter Hawkins, Mark S Miller, Franz Josef Och, Christopher Olston, and Fernando Pereira. Yedalog: Exploring Knowledge at Scale. In *Proceedings of SNAPL*, volume 32 of *LIPICs*, pages 63–78. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2015.
- [47] E F Codd. Further Normalization of the Data Base Relational Model. *Research Report / RJ / IBM / San Jose, California*, RJ909, 1971.
- [48] Stefano Concaro, Lucia Sacchi, Carlo Cerra, Pietro Fratino, and Riccardo Bellazzi. Mining Healthcare Data with Temporal Association Rules: Improvements and Assessment for a Practical Use. In *Proceedings of AIME*, volume 5651 of *Lecture Notes in Computer Science*, pages 16–25, 2009.
- [49] SQLite Consortium. SQLite.
- [50] Microsoft Corporation. Microsoft SQL Server.
- [51] Oracle Corporation. MySQL.
- [52] Oracle Corporation. Oracle Database.

- [53] Stavros S Cosmadakis, Haim Gaifman, Paris C Kanellakis, and Moshe Y Vardi. Decidable Optimization Problems for Database Logic Programs (Preliminary Report). In *Proceedings of ACM STOC*, pages 477–490. ACM, 1988.
- [54] Chad Creighton and Samir Hanash. Mining gene expression databases for association rules. *Bioinform.*, 19(1):79–86, 2003.
- [55] Evgeny Dantsin, Thomas Eiter, Georg Gottlob, and Andrei Voronkov. Complexity and expressive power of logic programming. *ACM Comput. Surv.*, 33(3):374–425, 2001.
- [56] Xin Dong, Evgeniy Gabrilovich, Jeremy Heitz, Wilko Horn, Ni Lao, Kevin Murphy, Thomas Strohmman, Shaohua Sun, and Wei Zhang. Knowledge vault: a web-scale approach to probabilistic knowledge fusion. In *Proceedings of ACM KDD*, pages 601–610. ACM, 2014.
- [57] Rafee Ebrahim. Fuzzy logic programming. *Fuzzy Sets Syst.*, 117(2):215–230, 2001.
- [58] Lisa Ehrlinger and Wolfram Wöß. Towards a Definition of Knowledge Graphs. In *SEMANTiCS (Posters, Demos, SuCCESS)*, 2016.
- [59] Jason Eisner and Nathaniel Wesley Filardo. Dyna: Extending Datalog for Modern AI. In *Proceedings of Datalog Reloaded: First International Workshop*, volume 6702 of *Lecture Notes in Computer Science*, pages 181–220. Springer, 2010.
- [60] Ronald Fagin. Horn clauses and database dependencies. *J. ACM*, 29(4):952–985, 1982.
- [61] 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.
- [62] Christiane Fellbaum. *WordNet: An electronic lexical database*. MIT press, 1998.
- [63] Sébastien Ferré. Link Prediction in Knowledge Graphs with Concepts of Nearest Neighbours. In *Proceedings of ESWC*, volume 11503 of *Lecture Notes in Computer Science*, pages 84–100. Springer, 2019.

- [64] Andreas Fjellstad and Jan-Fredrik Olsen. IKT^ω and Łukasiewicz-models. *Notre Dame Journal of Formal Logic*, 62(2):247–256, 2021.
- [65] Apache Software Foundation. Apache CouchDB.
- [66] The Linux Foundation. JanusGraph.
- [67] Eibe Frank, Mark A Hall, Geoffrey Holmes, Richard Kirkby, and Bernhard Pfahringer. WEKA - A Machine Learning Workbench for Data Mining. In *The Data Mining and Knowledge Discovery Handbook*, pages 1305–1314. Springer, 2005.
- [68] Norbert Fuhr. Probabilistic datalog: Implementing logical information retrieval for advanced applications. *J. Am. Soc. Inf. Sci.*, 51(2):95–110, 2000.
- [69] Johannes Fürnkranz and Tomás Kliegr. A Brief Overview of Rule Learning. In *Proceedings of RuleML*, volume 9202 of *Lecture Notes in Computer Science*, pages 54–69. Springer, 2015.
- [70] Enrique Garcia, Cristóbal Romero, Sebastián Ventura, and Carlos de Castro. A collaborative educational association rule mining tool. *Internet High. Educ.*, 14(2):77–88, 2011.
- [71] M Gergatsoulis. *Logic program transformations: Transformation rules and application strategies*. PhD thesis, PhD thesis, Dept. of Computer Science, University of Athens, 1994.(In Greek), 1994.
- [72] Manolis Gergatsoulis and Maria Katzouraki. Unfold/fold transformations for definite clause programs. In *Proceedings of PLILP*, pages 340–354. Springer, 1994.
- [73] Manisha Girotra, Kanika Nagpal, Saloni Minocha, and Neha Sharma. Comparative survey on association rule mining algorithms. *International Journal of Computer Applications*, 84(10), 2013.
- [74] James B Glattfelder. *Ownership networks and corporate control: mapping economic power in a globalized world*. PhD thesis, ETH Zurich, 2010.
- [75] Georg Gottlob, Jinsong Guo, Aditya Jami, Markus Kröll, Stéphane Reissfeldee, Lukas Schweizer, Eric Aichinger, and Stefano Sferrazza. CompeGen:

- Computing Company Competitor Pairs By Knowledge Based Inference Combined With Empirical Validation. In *Proceedings of Datalog-2.0*, volume 3203 of *CEUR Workshop Proceedings*, pages 207–211. CEUR-WS.org, 2022.
- [76] Georg Gottlob, Marco Manna, and Andreas Pieris. Polynomial Rewritings for Linear Existential Rules. In *Proceedings of IJCAI*, pages 2992–2998. AAAI Press, 2015.
 - [77] Georg Gottlob and Andreas Pieris. Beyond SPARQL under OWL 2 QL Entailment Regime: Rules to the Rescue. In *Proceedings of IJCAI*, pages 2999–3007. AAAI Press, 2015.
 - [78] Georg Gottlob, Sebastian Rudolph, and Mantas Simkus. Expressiveness of guarded existential rule languages. In *Proceedings of SIGMOD*, pages 27–38, 2014.
 - [79] Goetz Graefe and William J McKenna. The volcano optimizer generator: Extensibility and efficient search. In *Proceedings of ICDE*, pages 209–218, 1993.
 - [80] The PostgreSQL Global Development Group. PostgreSQL.
 - [81] Jinsong Guo, Aditya Jami, Markus Kröll, Lukas Schweizer, Sergey Paramonov, Eric Aichinger, Stefano Sferrazza, Mattia Scaccia, Stéphane Reissfelder, Eda Cicek, Giovanni Grasso, and Georg Gottlob. When Automatic Filtering Comes to the Rescue: Pre-Computing Company Competitor Pairs in Owler. *Proceedings of ACM Manag. Data*, 1(2):202:1–202:23, 2023.
 - [82] Gurobi Optimization, LLC. Gurobi Optimizer Reference Manual, 2021.
 - [83] Petr Hájek. *Metamathematics of Fuzzy Logic*, volume 4 of *Trends in Logic*. Kluwer, 1998.
 - [84] Jiawei Han, Micheline Kamber, and Jian Pei. *Data Mining: Concepts and Techniques, 3rd edition*. Morgan Kaufmann, 2011.
 - [85] Jiawei Han, Jian Pei, and Yiwen Yin. Mining Frequent Patterns without Candidate Generation. In *Proceedings of ACM SIGMOD*, pages 1–12. ACM, 2000.
 - [86] Sue-Chen Hsueh and Chia-Hsin Kuo. Effective Matching for P2P Lending by Mining Strong Association Rules. In *Proceedings of ICIBE*, pages 30–33. ACM, 2017.

- [87] ArangoDB Inc. ArangoDB.
- [88] Couchbase Inc. Couchbase.
- [89] MongoDB Inc. MongoDB.
- [90] Pascual Julián Iranzo and Fernando Sáenz-Pérez. A Fuzzy Datalog Deductive Database System. *IEEE Trans. Fuzzy Syst.*, 26(5):2634–2648, 2018.
- [91] Arpit Jain and Abhinav Sharma. Membership function formulation methods for fuzzy logic systems: A comprehensive review. *Proceedings of JCR*, 7(19):8717–8733, 2020.
- [92] George F Jenks. Optimal data classification for choropleth maps. *Department of Geography, University of Kansas Occasional Paper*, 1977.
- [93] Leonid Genrikhovich Khachiyan. A polynomial algorithm in linear programming. In *Doklady Akademii Nauk*, volume 244, pages 1093–1096. Russian Academy of Sciences, 1979.
- [94] George J Klir and Bo Yuan. *Fuzzy sets and fuzzy logic - theory and applications*. Prentice Hall, 1995.
- [95] Stanley Kok and Pedro M Domingos. Learning the structure of Markov logic networks. In *Proceedings of ICML*, volume 119 of *ACM International Conference Proceeding Series*, pages 441–448. ACM, 2005.
- [96] Markus Krötzsch and Sebastian Rudolph. Extending Decidable Existential Rules by Joining Acyclicity and Guardedness. In *Proceedings of IJCAI*, pages 963–968. IJCAI/AAAI, 2011.
- [97] Trupti A Kumbhare and Santosh V Chobe. An overview of association rule mining algorithms. *Proceedings of IJCSIT*, 5(1):927–930, 2014.
- [98] Niels Landwehr, Kristian Kersting, and Luc De Raedt. Integrating Naive Bayes and FOIL. *J. Mach. Learn. Res.*, 8:481–507, 2007.
- [99] Niels Landwehr, Andrea Passerini, Luc De Raedt, and Paolo Frasconi. Fast learning of relational kernels. *Mach. Learn.*, 78(3):305–342, 2010.
- [100] Shaily G Langhnoja, Mehul P Barot, and Darshak B Mehta. Web usage mining using association rule mining on clustered data for pattern discovery. *International Journal of Data Mining Techniques and Applications*, 2(1):141–150, 2013.

- [101] Matthias Lanzinger, Stefano Sferrazza, and Georg Gottlob. MV-Datalog+-: Effective Rule-based Reasoning with Uncertain Observations, 2022.
- [102] Matthias Lanzinger, Stefano Sferrazza, and Georg Gottlob. New perspectives for fuzzy datalog (extended abstract). In Mario Alviano and Andreas Pieris, editors, *Proceedings of Datalog-2.0*, volume 3203 of *CEUR Workshop Proceedings*, pages 42–47. CEUR-WS.org, 2022.
- [103] Han S Lee, Heechul Jung, Alex A Agarwal, and Junmo Kim. Can Deep Neural Networks Match the Related Objects?: A Survey on ImageNet-trained Classification Models. *CoRR*, abs/1709.0, 2017.
- [104] Jens Lehmann, Robert Isele, Max Jakob, Anja Jentzsch, Dimitris Kontokostas, Pablo N Mendes, Sebastian Hellmann, Mohamed Morsey, Patrick van Kleef, Sören Auer, and Christian Bizer. DBpedia - A large-scale, multilingual knowledge base extracted from Wikipedia. *Semantic Web*, 6(2):167–195, 2015.
- [105] Nicola Leone, Marco Manna, Giorgio Terracina, and Pierfrancesco Veltri. DLV^E System. <https://www.mat.unical.it/dlve/>, 2017.
- [106] Vladimir I Levenshtein. Binary codes capable of correcting deletions, insertions, and reversals. In *Soviet Physics Doklad*, volume 10 (8), pages 707–710, 1966.
- [107] Hibernating Rhinos Ltd. RavenDB.
- [108] OrientDB Ltd. OrientDB.
- [109] Thomas Lukasiewicz and Umberto Straccia. Managing uncertainty and vagueness in description logics for the Semantic Web. *J. Web Semant.*, 6(4):291–308, 2008.
- [110] Jianxiong Luo. *Integrating fuzzy logic with data mining methods for intrusion detection*. Mississippi State University, 1999.
- [111] David Maier, Alberto O Mendelzon, and Yehoshua Sagiv. Testing implications of data dependencies. *Proceedings of ACM TODS*, 4(4):455–469, 1979.
- [112] David Maier, Yehoshua Sagiv, and Mihalys Yannakakis. On the Complexity of Testing Implications of Functional and Join Dependencies. *J. ACM*, 28(4):680–695, 1981.

- [113] David Maier, K Tuncay Tekle, Michael Kifer, and David Scott Warren. Datalog: concepts, history, and outlook. In *Declarative Logic Programming: Theory, Systems, and Applications*. ACM / Morgan & Claypool, 2018.
- [114] Robin Manhaeve, Sebastijan Dumancic, Angelika Kimmig, Thomas De-meester, and Luc De Raedt. Neural probabilistic logic programming in DeepProbLog. *Artif. Intell.*, 298:103504, 2021.
- [115] Bruno Marnette. Generalized schema-mappings: from termination to tractability. In *Proceedings of PODS*, pages 13–22. ACM, 2009.
- [116] Victor Martinez, Fernando Berzal, and Juan Carlos Cubero Talavera. A Survey of Link Prediction in Complex Networks. *ACM Comput. Surv.*, 49(4):69:1–69:33, 2017.
- [117] Christian Meilicke, Melisachew Wudage Chekol, Daniel Ruffinelli, and Heiner Stuckenschmidt. Anytime Bottom-Up Rule Learning for Knowledge Graph Completion. In *Proceedings of IJCAI*, pages 3137–3143. ijcai.org, 2019.
- [118] Christian Meilicke, Manuel Fink, Yanjie Wang, Daniel Ruffinelli, Rainer Gemulla, and Heiner Stuckenschmidt. Fine-Grained Evaluation of Rule- and Embedding-Based Systems for Knowledge Graph Completion. In *Proceedings of ISWC*, volume 11136 of *Lecture Notes in Computer Science*, pages 3–20. Springer, 2018.
- [119] Meltwater. Meltwater: Media Monitoring and Social Media Monitoring, 2024.
- [120] George A Miller. WordNet: A Lexical Database for English. *Commun. ACM*, 38(11):39–41, 1995.
- [121] Vuqar Mirzakhanov. Value of fuzzy logic for data mining and machine learning: A case study. *Expert Syst. Appl.*, 162:113781, 2020.
- [122] Dmytro Mishkin, Nikolay Sergievskiy, and Jiri Matas. Systematic evaluation of convolution neural network advances on the Imagenet. *Comput. Vis. Image Underst.*, 161:11–19, 2017.
- [123] Stephen Muggleton. Learning Stochastic Logic Programs. *Electron. Trans. Artif. Intell.*, 4(B):141–153, 2000.

- [124] Stephen H Muggleton and Luc De Raedt. Inductive Logic Programming: Theory and Methods. *J. Log. Program.*, 19/20:629–679, 1994.
- [125] Stefan Naulaerts, Pieter Meysman, Wout Bittremieux, Trung-Nghia Vu, Wim Vanden Berghe, Bart Goethals, and Kris Laukens. A primer to frequent itemset mining for bioinformatics. *Briefings Bioinform.*, 16(2):216–231, 2015.
- [126] Roberto Navigli and Simone Paolo Ponzetto. BabelNet: Building a Very Large Multilingual Semantic Network. In *Proceedings of ACL*, pages 216–225. The Association for Computer Linguistics, 2010.
- [127] Radford M Neal. Probabilistic inference using Markov chain Monte Carlo methods. 1993.
- [128] Yavor Nenov, Robert Piro, Boris Motik, Ian Horrocks, Zhe Wu, and Jay Banerjee. RDFox: A highly-scalable RDF store. In *Proceedings of ISWC*, volume 9367 of *Lecture Notes in Computer Science*, pages 3–20. Springer, 2015.
- [129] Simon Ott, Christian Meilicke, and Matthias Samwald. SAFRAN: An interpretable, rule-based link prediction method outperforming embedding models. In *Proceedings of AKBC*, 2021.
- [130] Owler. Owler: Competitive Intelligence to Outsmart Your Competition, 2024.
- [131] Peter F Patel-Schneider and Ian Horrocks. Position paper: a comparison of two modelling paradigms in the Semantic Web. In *Proceedings of WWW*, pages 3–12. ACM, 2006.
- [132] Judea Pearl. *Probabilistic reasoning in intelligent systems - networks of plausible inference*. Morgan Kaufmann series in representation and reasoning. Morgan Kaufmann, 1989.
- [133] Alberto Pettorossi and Maurizio Proietti. Transformation of logic programs: Foundations and techniques. *Proceedings of J. Log. Program*, 19:261–320, 1994.
- [134] David Poole. Probabilistic Horn Abduction and Bayesian Networks. *Artif. Intell.*, 64(1):81–129, 1993.

- [135] Norbert Preining. Gödel Logics - A Survey. In *Proceedings of LPAR*, volume 6397 of *Lecture Notes in Computer Science*, pages 30–51. Springer, 2010.
- [136] Maurizio Proietti and Alberto Pettorossi. The loop absorption and the generalization strategies for the development of logic programs and partial deduction. *Proceedings of J. Log. Program*, 16(1-2):123–161, 1993.
- [137] Luc De Raedt, Angelika Kimmig, and Hannu Toivonen. ProbLog: A Probabilistic Prolog and Its Application in Link Discovery. In *Proceedings of IJCAI*, pages 2462–2467, 2007.
- [138] Raghu Ramakrishnan, Catriel Beeri, and Ravi Krishnamurthy. Optimizing Existential Datalog Queries. In *Proceedings of ACM PODS*, pages 89–102. ACM, 1988.
- [139] Matthew Richardson and Pedro M Domingos. Markov logic networks. *Mach. Learn.*, 62(1-2):107–136, 2006.
- [140] Lior Rokach. Using Fuzzy Logic in Data Mining. In *Data Mining and Knowledge Discovery Handbook, 2nd ed*, pages 505–520. Springer, 2010.
- [141] Andrea Romei, Salvatore Ruggieri, and Franco Turini. The layered structure of company share networks. In *Proceedings of DSAA*, pages 1–10. IEEE, 2015.
- [142] Cristóbal Romero, José Raúl Romero, José Maria Luna, and Sebastián Ventura. Mining Rare Association Rules from e-Learning Data. In *Proceedings of EDM*, pages 171–180. www.educationaldatamining.org, 2010.
- [143] Timothy Roscoe and Boon Thau Loo. Declarative Networking. In *Encyclopedia of Database Systems, Second Edition*. Springer, 2018.
- [144] Ayse Nur Sagin and Berk Ayvaz. Determination of association rules with market basket analysis: application in the retail sector. *Proceedings of SEJSC*, 7(1), 2018.
- [145] Yatin P Saraiya. On the Efficiency of Transforming Database Logic Programs. *J. Comput. Syst. Sci.*, 51(1):87–109, 1995.
- [146] Md. Kamruzzaman Sarker, Lu Zhou, Aaron Eberhart, and Pascal Hitzler. Neuro-symbolic artificial intelligence. *AI Commun.*, 34(3):197–209, 2021.

- [147] Taisuke Sato and Yoshitaka Kameya. Parameter Learning of Logic Programs for Symbolic-Statistical Modeling. *Proceedings of JAIR*, 15:391–454, 2001.
- [148] 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.
- [149] Amazon Web Services. Amazon DocumentDB.
- [150] Amazon Web Services. Amazon Neptune.
- [151] Stefano Sferrazza, Georg Gottlob, Giovanni Grasso, Aditya Jami, Markus Kröll, and Lukas Schweizer. Association-rule based data outlier detection and record completion system and method, 2023.
- [152] Alexander Shkapsky, Mohan Yang, and Carlo Zaniolo. Optimizing recursive queries with monotonic aggregates in DeALS. In *Proceedings of ICDE*, pages 867–878, 2015.
- [153] Arnab Sinha, Zhihong Shen, Yang Song, Hao Ma, Darrin Eide, Bo-June Paul Hsu, and Kuansan Wang. An Overview of Microsoft Academic Service (MAS) and Applications. In *Proceedings of WWW*, pages 243–246. ACM, 2015.
- [154] Robyn Speer, Joshua Chin, and Catherine Havasi. ConceptNet 5.5: An Open Multilingual Graph of General Knowledge. In *Proceedings of AAAI*, pages 4444–4451. AAAI Press, 2017.
- [155] Statista. Number of companies worldwide 2000-2021, 2023.
- [156] S Stilou, Panagiotis D Bamidis, Nicos Maglaveras, and Constantinos Pappas. Mining Association Rules from Clinical Databases: An Intelligent Diagnostic Process in Healthcare. In *Proceedings of MEDINFO*, volume 84 of *Studies in Health Technology and Informatics*, pages 1399–1403. IOS Press, 2001.
- [157] Giorgos Stoilos, Giorgos B Stamou, Jeff Z Pan, Vassilis Tzouvaras, and Ian Horrocks. Reasoning with Very Expressive Fuzzy Description Logics. *Proceedings of JAIR*, 30:273–320, 2007.
- [158] Fabian M Suchanek, Gjergji Kasneci, and Gerhard Weikum. Yago: a core of semantic knowledge. In *Proceedings of WWW*, pages 697–706. ACM, 2007.
- [159] Dan Suciu, Dan Olteanu, Christopher Ré, and Christoph Koch. *Probabilistic Databases*. Synthesis Lectures on Data Management. Morgan & Claypool Publishers, 2011.

- [160] Mingxing Tan and Quoc V Le. EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks. In *Proceedings of ICML*, volume 97 of *Proceedings of Machine Learning Research*, pages 6105–6114. PMLR, 2019.
- [161] Neo Technology. Neo4j.
- [162] Efthymia Tsamoura, Timothy M Hospedales, and Loizos Michael. Neural-Symbolic Integration: A Compositional Perspective. In *Proceedings of AAAI*, pages 5051–5060. AAAI Press, 2021.
- [163] Leslie G Valiant. The Complexity of Enumeration and Reliability Problems. *SIAM J. Comput.*, 8(3):410–421, 1979.
- [164] Victor Vianu. Datalog Unchained. In *Proceedings of ACM PODS*, pages 57–69, 2021.
- [165] Peter Vojtás. Fuzzy logic programming. *Fuzzy Sets Syst.*, 124(3):361–370, 2001.
- [166] Denny Vrandečić and Markus Krötzsch. Wikidata: a free collaborative knowledgebase. *Commun. ACM*, 57(10):78–85, 2014.
- [167] Hongzhi Wang, Mohamed Jaward Bah, and Mohamed Hammad. Progress in Outlier Detection Techniques: A Survey. *IEEE Access*, 7:107964–108000, 2019.
- [168] Jingjing Wang, Magdalena Balazinska, and Daniel Halperin. Asynchronous and Fault-Tolerant Recursive Datalog Evaluation in Shared-Nothing Engines. *Proceedings of VLDB Endow.*, 8(12):1542–1553, 2015.
- [169] Wikipedia. Owl — Wikipedia, The Free Encyclopedia, 2023.
- [170] Wentao Wu, Hongsong Li, Haixun Wang, and Kenny Qili Zhu. Probase: a probabilistic taxonomy for text understanding. In *Proceedings of ACM SIGMOD*, pages 481–492. ACM, 2012.
- [171] Jonathan S Yedidia, William T Freeman, Yair Weiss, and others. Understanding belief propagation and its generalizations. *Exploring artificial intelligence in the new millennium*, 8(236-239):18–9448, 2003.
- [172] Lotfi A Zadeh. Fuzzy Sets. *Inf. Control.*, 8(3):338–353, 1965.

- [173] Mohammed Javeed Zaki. Scalable Algorithms for Association Mining. *IEEE Trans. Knowl. Data Eng.*, 12(3):372–390, 2000.
- [174] Qiang Zeng, Jignesh M Patel, and David Page. QuickFOIL: Scalable Inductive Logic Programming. *Proceedings of VLDB Endow.*, 8(3):197–208, 2014.
- [175] Zefan Zeng, Qing Cheng, and Yuehang Si. Logical Rule-Based Knowledge Graph Reasoning: A Comprehensive Survey. *Mathematics*, 11(21):4486, 2023.

Appendix

Additional Figures of $ARM(D, I)$

In this section, we include additional figures for Section 5.1:

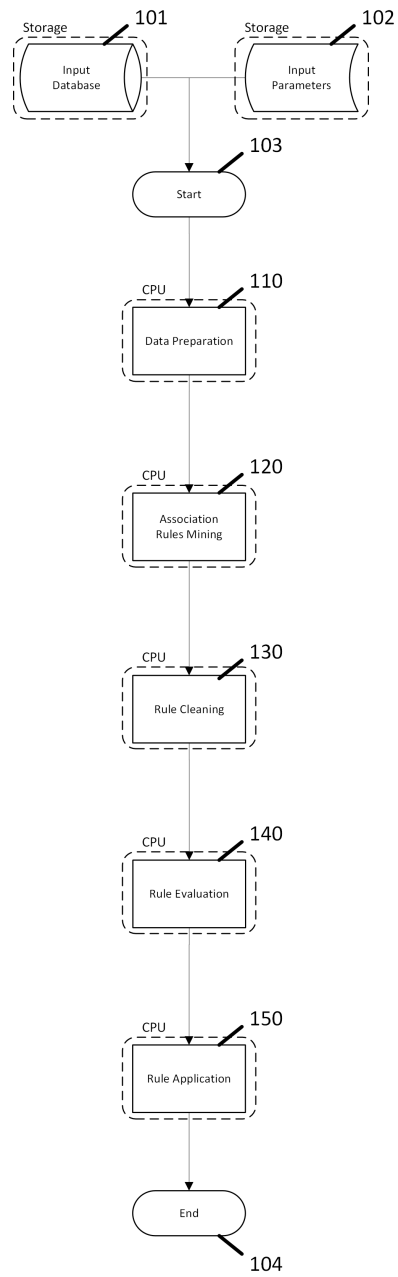


Figure A.1: This figure depicts a subset of the flowchart shown in Figure 5.1 with a focus on the main processes.

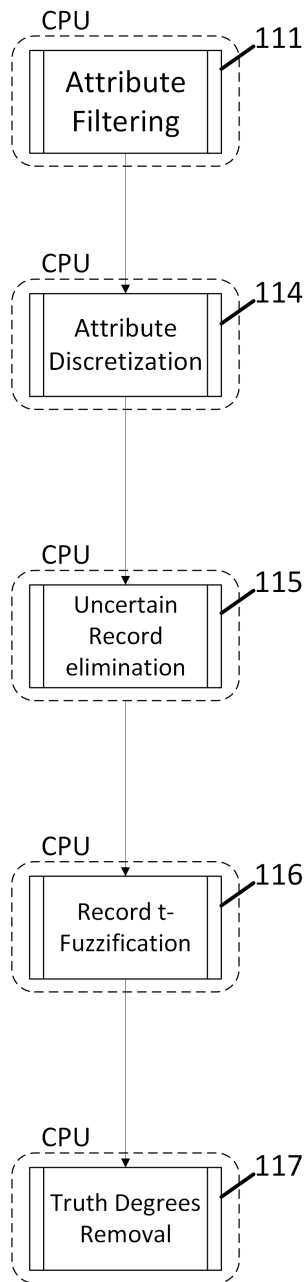


Figure A.2: This figure depicts a subset of the flowchart shown in Figure 5.1 with a focus on the subprocesses of **110**.

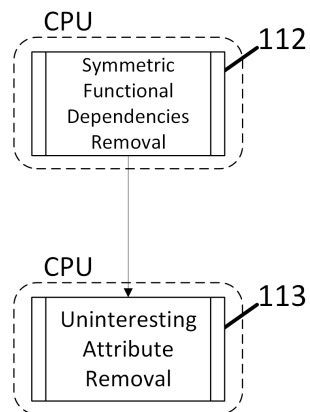


Figure A.3: This figure depicts a subset of the flowchart shown in Figure 5.1 with a focus on the subprocesses of **111**.

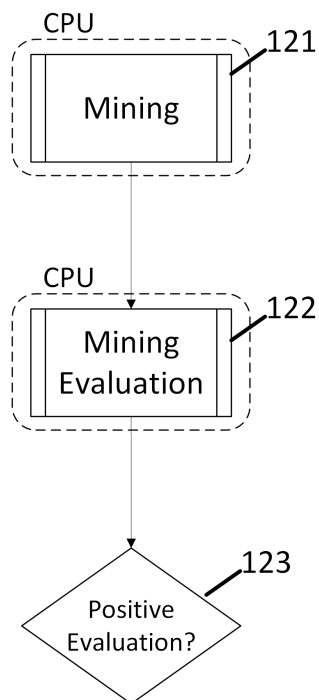


Figure A.4: This figure depicts a subset of the flowchart shown in Figure 5.1 with a focus on the subprocesses of **120**.

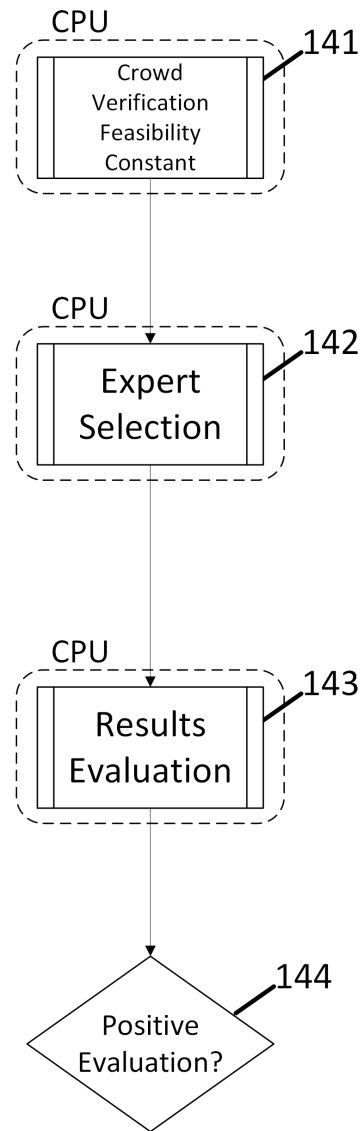


Figure A.5: This figure depicts a subset of the flowchart shown in Figure 5.1 with a focus on the subprocesses of **140**.

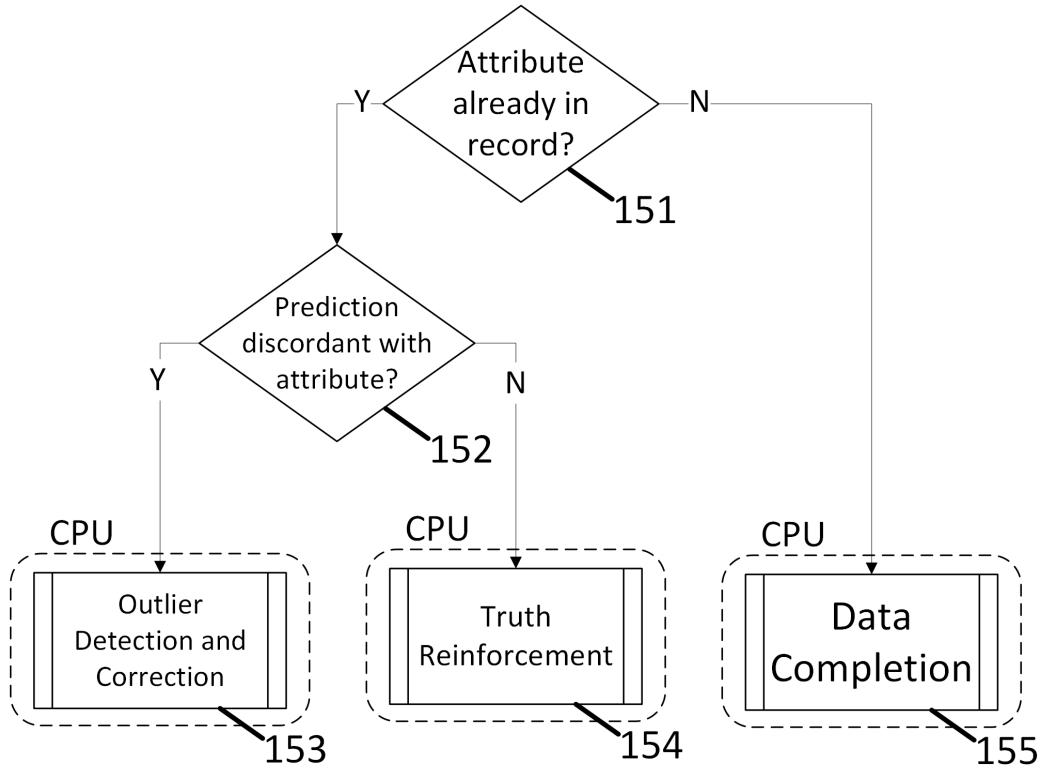


Figure A.6: This figure depicts a subset of the flowchart shown in Figure 5.1 with a focus on the subprocesses of **150**.

Additional Association Rules

Here, we include more association rules mined during the experiments described in Section 5.2. First, we list a set of 20 association rules to predict values for the *revenue* attribute.

$$\begin{aligned}
 &employees(Company_ID, "10000 < Emp"), \\
 &\quad founded(Company_ID, "20 < age") \\
 &\rightarrow revenue(Company_ID, "50M \leq Rev").
 \end{aligned}$$

$$\begin{aligned}
 &country(Company_ID, "USA"), employees(Company_ID, "10000 < Emp") \\
 &\rightarrow revenue(Company_ID, "50M \leq Rev").
 \end{aligned}$$

exchange(Company_ID, "NYSE"), employees(Company_ID, "10000 < Emp")
→ revenue(Company_ID, "50M <= Rev").

employees(Company_ID, "10000 < Emp")
→ revenue(Company_ID, "50M <= Rev").

country(Company_ID, "USA"), status(Company_ID, "Public"),
employees(Company_ID, "1000 < Emp <= 10000")
→ revenue(Company_ID, "50M <= Rev").

exchange(Company_ID, "NASDAQ"),
employees(Company_ID, "1000 < Emp <= 10000")
→ revenue(Company_ID, "50M <= Rev").

country(Company_ID, "USA"),
employees(Company_ID, "1000 < Emp <= 10000")
→ revenue(Company_ID, "50M <= Rev").

country(Company_ID, "USA"), exchange(Company_ID, "NYSE"),
employees(Company_ID, "1000 < Emp <= 10000")
→ revenue(Company_ID, "50M <= Rev").

exchange(Company_ID, "NYSE"),
employees(Company_ID, "1000 < Emp <= 10000"),
founded(Company_ID, "20 < age")
→ revenue(Company_ID, "50M <= Rev").

employees(Company_ID, "1000 < Emp <= 10000")
→ revenue(Company_ID, "50M <= Rev").

country(Company_ID, "USA"), status(Company_ID, "Public"),
exchange(Company_ID, "NYSE"), employees(Company_ID, "100 < Emp"),
founded(Company_ID, "20 < age")
→ revenue(Company_ID, "50M <= Rev").

country(Company_ID, "USA"), status(Company_ID, "Public"),
exchange(Company_ID, "NYSE"),
employees(Company_ID, "100 < Emp")
→ revenue(Company_ID, "50M <= Rev").

status(Company_ID, "Public"), exchange(Company_ID, "NYSE"),
employees(Company_ID, "100 < Emp"),
founded(Company_ID, "20 < age")
→ revenue(Company_ID, "50M <= Rev").

status(Company_ID, "Private"),
employees(Company_ID, "1000 < Emp <= 10000"),
founded(Company_ID, "20 < age")
→ revenue(Company_ID, "50M <= Rev").

status(Company_ID, "Public"), exchange(Company_ID, "NYSE"),
employees(Company_ID, "100 < Emp")
→ revenue(Company_ID, "50M <= Rev").

status(Company_ID, "Private"),
employees(Company_ID, "1000 < Emp <= 10000")
→ revenue(Company_ID, "50M <= Rev").

country(Company_ID, "USA"), status(Company_ID, "Public"),
exchange(Company_ID, "NYSE"), founded(Company_ID, "20 < age")
→ revenue(Company_ID, "50M <= Rev").

country(Company_ID, "USA"),
employees(Company_ID, "10000 < Emp")
→ revenue(Company_ID, "50M <= Rev").

status(Company_ID, "Public"), exchange(Company_ID, "NYSE"),
founded(Company_ID, "20 < age")
→ revenue(Company_ID, "50M <= Rev").

country(Company_ID, "USA"), status(Company_ID, "Public"),
exchange(Company_ID, "NASDAQ"), employees(Company_ID, "100 < Emp"),
founded(Company_ID, "20 < age")
→ revenue(Company_ID, "50M <= Rev").

In the following, we list a set of 20 association rules to predict values for the *employees* attribute.

revenue(Company_ID, "1B <= Rev < 10B"),
founded(Company_ID, "20 < age")
→ employees(Company_ID, "100 < Emp").

country(Company_ID, "USA"),
revenue(Company_ID, "1B <= Rev < 10B")
→ employees(Company_ID, "100 < Emp").

country(Company_ID, "USA"), exchange(Company_ID, "NYSE"),
revenue(Company_ID, "1B <= Rev < 10B") →
employees(Company_ID, "100 < Emp").

exchange(Company_ID, "NYSE"),
revenue(Company_ID, "1B <= Rev < 10B"),
founded(Company_ID, "20 < age")
→ employees(Company_ID, "100 < Emp").

country(Company_ID, "USA"), status(Company_ID, "Public"),
revenue(Company_ID, "1B <= Rev < 10B"),
founded(Company_ID, "20 < age")
→ employees(Company_ID, "100 < Emp").

revenue(Company_ID, "10B <= Rev < 50B")
→ employees(Company_ID, "100 < Emp").

revenue(Company_ID, "1B <= Rev < 10B")
→ employees(Company_ID, "100 < Emp").

industry(Company_ID, "Software, Internet&ComputerServices"),
revenue(Company_ID, "50M <= Rev"),
founded(Company_ID, "20 < age")
→ employees(Company_ID, "100 < Emp").

industry(Company_ID, "Software, Internet&ComputerServices"),
revenue(Company_ID, "100M <= Rev < 1B")
→ employees(Company_ID, "100 < Emp").

status(Company_ID, "Public"), exchange(Company_ID, "NYSE"),
revenue(Company_ID, "1B <= Rev < 10B")
→ employees(Company_ID, "100 < Emp").

status(Company_ID, "Public"),
industry(Company_ID, "Software, Internet&ComputerServices"),
revenue(Company_ID, "50M <= Rev")
→ employees(Company_ID, "100 < Emp").

industry(Company_ID, "Software, Internet&ComputerServices"),
revenue(Company_ID, "50M <= Rev")
→ employees(Company_ID, "100 < Emp").

country(Company_ID, "USA"), status(Company_ID, "Public"),
exchange(Company_ID, "NYSE"),
revenue(Company_ID, "50M <= Rev"),
founded(Company_ID, "20 < age")
→ employees(Company_ID, "100 < Emp").

exchange(Company_ID, "NASDAQ"),
revenue(Company_ID, "100M <= Rev < 1B"),
founded(Company_ID, "20 < age")
→ employees(Company_ID, "100 < Emp").

exchange(Company_ID, "NYSE"), revenue(Company_ID, "50M <= Rev"),
founded(Company_ID, "20 < age")
→ employees(Company_ID, "100 < Emp").

country(Company_ID, "USA"), exchange(Company_ID, "NYSE"),
revenue(Company_ID, "50M <= Rev"), founded(Company_ID, "20 < age")
→ employees(Company_ID, "100 < Emp").

status(Company_ID, "Public"), exchange(Company_ID, "NYSE"),
revenue(Company_ID, "50M <= Rev"), founded(Company_ID, "20 < age")
→ employees(Company_ID, "100 < Emp").

revenue(Company_ID, "50M <= Rev"), founded(Company_ID, "20 < age1")
→ employees(Company_ID, "100 < Emp").

status(Company_ID, "Public"), revenue(Company_ID, "50M <= Rev"),
founded(Company_ID, "20 < age")
→ employees(Company_ID, "100 < Emp").

country(Company_ID, "USA"), revenue(Company_ID, "50M <= Rev"),
founded(Company_ID, "20 < age")
→ employees(Company_ID, "100 < Emp").