Defeasible DLV Literature Review

Tala Ross
rsstal002@myuct.ac.za
University of Cape Town
Cape Town, South Africa

ABSTRACT

Datalog is a declarative logic programming language that uses classical logical reasoning as its basic form of reasoning. DLV is a disjunctive logic programming language, which forms an extension of Datalog. Defeasible reasoning is a form of non-classical reasoning that is able to deal with exceptions to general assertions in a formal manner. Since Datalog and DLV use classical reasoning, they are currently not able to handle defeasible implications and exceptions. We aim to extend the expressivity of Datalog by incorporating KLM-style[20] defeasible reasoning into classical Datalog and extending DLV with defeasible reasoning.

CCS CONCEPTS

• **Theory of computation → Automated reasoning**; *Logic and databases*; Database query languages (principles); • **Computing methodologies → Nonmonotonic, default reasoning and belief revision**; *Description logics*;

KEYWORDS

Artificial Intelligence, Knowledge Representation and Reasoning, classical reasoning, defeasible reasoning, propositional logic, first-order logic, description logics, disjunctive logic, declarative logic programming, Datalog, DLV

1 INTRODUCTION

There are two main approaches to Artificial Intelligence, Machine Learning (ML) and Knowledge Representation and Reasoning(KRR). We will focus our attention on the KRR approach. Knowledge representation is the use of symbols to stand for knowledge about some problem domain. Reasoning is the manipulation of symbols which encode propositions to produce new propositions. We use different logics to represent knowledge and we use automated reasoning methods to reason about that knowledge.

A *logic*, or *logical system*, is a mechanism which formalizes valid ways to reason.[2] It has a formal *language* for making statements about objects and reasoning about properties of these objects. The *syntax* of the language, given by a predefined set of rules, defines the legal structure of statements in the language. The *semantics* of the language is the meaning of statements, defined by *interpretations*, which assign a single *truth value*, either *true* or *false*, to a statement.

This is a very general definition of logic and there are, in fact, many different types of logics. In Section 2 we will focus our attention on *Propositional logic*, a simple logic, forming the basis of many other logics. Then, in Section 3, we will briefly turn our attention to *First-order logic*, an extension of Propositional logic, and the family of Description logics which are fragments of First-order logic.

In Section 4, we then briefly describe the concepts of Classical Reasoning and Tarskian operators which satisfy the monotonicity property. Then, in Section 5, we describe Defeasible reasoning, a form of non-monotonic "common-sense" reasoning which allows for exceptions. We focus mainly on the KLM[20] approach and briefly contrast this to a few other possible approaches.

Next, in Section 6, we describe the syntax and semantics of Datalog and provide motivations for its use. Finally, in Section 7, we describe the extension of the syntax and semantics of Datalog to include disjunction, which forms the language of DLV. We then explore a possible approach to extending DLV to handle defeasible reasoning based on the concepts presented in the previous sections. Lastly, we will conclude about the motivations for and feasibility of extending Datalog and DLV using Defeasible reasoning.

2 PROPOSITIONAL LOGIC

As humans we make statements such as “Birds can fly” and “Tweety is a bird” and we deduce intuitively that Tweety can fly. However, without a formalization of how to state assertions and how to deduce information from assertions, we have no way to analyse how appropriate or correct our reasoning is. We say that an *atom*, or *atomic proposition*, is a statement that can be assigned a truth value (true or false), which cannot be decomposed into smaller such statements.[2] A propositional logic system is a logical system in which formulas can be formed by combining atomic propositions using Boolean operators. These formulas provide a formal way to represent assertions. In this section we will explore the syntax, semantics and pragmatics of propositional logic, as defined in [13] [2].

2.1 Syntax

To define the syntax for propositional logic, we need to define its alphabet and language. The alphabet is used to construct each formula. The language is the set of all possible formulas.

The *alphabet* for formulae consists of a countable set $\mathcal{P}$ of atoms denoted by $p, q, \ldots$, and, the *Boolean operators* $\neg$ (negation), $\wedge$ (conjunction), $\vee$ (disjunction), $\rightarrow$ (implication) and $\leftrightarrow$ (bi-implication), where $\neg$ is a unary operator and the other operators are binary operators.

A *formula* is a string over the alphabet $\mathcal{P}$, denoted by $\alpha, \beta, \ldots$. Representations of formulas as strings can be ambiguous. We use parentheses to preserve structure, and, a precedence order and associativity rules to resolve ambiguity. The precedence of operations from high to low is: $\neg, \wedge, \vee, \rightarrow, \leftrightarrow$.

The language $\mathcal{L}$ of propositional logic is the set of all formulae, defined recursively as follows:

- If p is an atom in $\mathcal{P}$, p is a formula in $\mathcal{L}$.
- If α and β are formulae in $\mathcal{L}$, then $\neg\alpha, \alpha \wedge \beta, \alpha \vee \beta, \alpha \rightarrow \beta$ and $\alpha \leftrightarrow \beta$ are all formulae in $\mathcal{L}$.

2.2 Semantics

To define the semantics of propositional logic we first define the meaning of the Boolean operators and then we use this to assign truth values to each formula.

Each Boolean operator is a function with domain $\mathcal{L}$ and range $\{T, F\}$, where $\neg$ is a function on a single variable and the binary operators are functions on two variables. The meaning of these functions is shown in the truth table in Table 1.

p	q	$\neg p$	$p \wedge q$	$p \vee q$	$p \rightarrow q$	$p \leftrightarrow q$
T	T	F	T	T	T	T
T	F	F	F	T	F	F
F	T	T	F	T	T	F
F	F	T	F	F	T	T

Figure 1: A truth table defining Boolean operators semantics

To formalize the ideas demonstrated in truth tables we define an interpretation function. An *interpretation* is a function $I : \mathcal{P} \rightarrow \{T, F\}$ which assigns a single truth value to each atom. We denote the set of interpretations by W. Intuitively, W is just the set of rows from a truth table for all atoms. Notice that since each atom can be assigned 2 possible values, true or false, there must be $2^{|\mathcal{P}|}$ possible interpretations in W.

Now we define satisfaction, a way of extending interpretations to each formula in $\mathcal{L}$. We say that an interpretation I *satisfies* a formula α, denoted $I \Vdash \alpha$, if and only if one of the following conditions holds:

- $\alpha \in \mathcal{P}$ and $I(\alpha) = T$
- $\alpha = \neg\beta$ and I does not satisfy β
- $\alpha = \beta \wedge \gamma$ and both $I \Vdash \beta$ and $I \Vdash \gamma$
- $\alpha = \beta \vee \gamma$ and at least one of $I \Vdash \beta$ or $I \Vdash \gamma$
- $\alpha = \beta \rightarrow \gamma$ and at least one of $I \Vdash \neg\beta$ or $I \Vdash \gamma$
- $\alpha = \beta \leftrightarrow \gamma$ and either $I \Vdash \beta$ and $I \Vdash \gamma$, or $I \Vdash \neg\beta$ and $I \Vdash \neg\gamma$

A *knowledge base* K is a finite set of formulas from $\mathcal{L}$. We say that α is *satisfiable* if there is an interpretation $I \in W$ such that $I \Vdash \alpha$; otherwise α is *unsatisfiable*. More generally, K is *satisfiable* if there is an interpretation $I \in W$ that satisfies all formulas in K.

The *models* of α, denoted $[\![\alpha]\!]$, is the set of interpretations that satisfy α. More generally, we define $[\![K]\!] = \{[\![\alpha]\!] : \alpha \in K\}$. We say α *entails* β, or β is a *logical consequence* of α, denoted $\alpha \Vdash \beta$, if and only if $[\![\beta]\!] \subseteq [\![\alpha]\!]$. That is, every interpretation that satisfies β also satisfies α. More generally, $K \Vdash \beta$ if and only if $[\![\beta]\!] \subseteq [\![K]\!]$.

3 EXTENDING PROPOSITIONAL LOGIC

Propositional logic only allows us to express knowledge about objects as a whole and not about their individual elements. For example, we may want to express that "Some birds are penguins" but we can only express that "All birds are penguins" using propositional logic. Thus, we often require more expressive forms of logic.

3.1 First-order Logic

First-order logic allows assertions about elements of structures to be expressed. This is achieved by extending Propositional logic to allow the propositional symbols to have arguments ranging over elements of structures. The syntax and semantics of first-order logic is essentially the syntax and semantics of formal mathematics. We will now very briefly explore this formalization, as defined in [13].

3.1.1 Syntax of First-order Logic. The *alphabet* of a First-order language is made up of a fixed part and a *non-logical part* whose contents depend on the intended application of the language. The fixed part consists of the Boolean operators of Propositional logic, the *quantifiers* $\forall$ (for all) and $\exists$ (there exists), the equality symbol = and a countably infinite set of *variables* denoted by $X, Y, Z, \ldots$. The non-logical part consists of a countable set of *functions* denoted by $f, g, h, \ldots$, a countable set of *constants* denoted by $a, b, c, \ldots$ and a countable set of *predicate symbols* denoted by $p, q, \ldots$. We then inductively use this alphabet to build terms and formulas, of the form familiar from formal mathematics, in order to construct the language of First-order logic.

3.1.2 Semantics of First-order Logic. To define semantics, the notion of a *structure* $S = (M, I)$, where M is a nonempty set called the *domain* and I is the *interpretation function*, needs to be introduced. This is essentially the notion of an algebra in formal mathematics. Then, we define entailment recursively on the set of formulas, such that, given a structure S and a formula α, for any assignment s of values in S to the variables in α, $S \Vdash \alpha[s]$. We say a formula is *valid* if it is valid in every structure.

3.2 Description Logics

Description Logics are a family of fragments of first-order logic. They can be used to represent declarative knowledge about a domain, while allowing automated reasoning with the knowledge to infer implicit facts about the domain.[6] Each Description Logic has a set of logical features which together determine its expressive power but the more expressive it is the more computationally intensive it is to perform automated reasoning with it.[6]

4 CLASSICAL REASONING

Classical reasoning is a reasoning framework in which new information is inferred from given information by making use of Tarskian consequence operators.

4.1 Tarskian Operators

A *consequence operator* is an operator which maps arbitrary sets of formulas of $\mathcal{L}$ to sets of formulas of $\mathcal{L}$, where $\mathcal{L}$ is some logical language. A consequence operator C_n is Tarskian[19] if the following properties hold for some set A:

- *Inclusion:* $A \subseteq C_n(A)$
- *Closure:* $C_n(C_n(A)) \subseteq C_n(A)$
- *Monotony:* If $A \subseteq B$ then $C_n(A) \subseteq C_n(B)$

Consider $Th(A) ::= \{\alpha \in \mathcal{L} : A \Vdash \alpha\}$ to be the set of all formulas in $\mathcal{L}$ which are a logical consequences of formulas in A. Notice that Th, in propositional and first-order logic and fragments is a Tarskian operator.

4.2 Shortfalls of Classical Reasoning

Classical reasoning cannot accommodate the addition of new information which contradicts what is known, since then learning a new piece of knowledge could possibly reduce the set of what is known, contradicting monotonicity. Consider the following example from [20]: if a monotonic system is told that “Penguins are bird” and “Birds fly” then when it encounters new information, an exception, that “Tweety is a Penguin”, it will still conclude that Tweety flies. To try handle this we can add the fact that “Penguins don’t fly” but this doesn’t work. Now we must conclude that no penguins exist, changing our mind about previously known information.

However, humans use “common-sense” reasoning, with exceptions, all the time. We make realistic conclusions from what we know and when presented with new information which contradicts our assumption we take back our previous conclusions but don’t change our minds about previous information. In this manner, in the previous example, we add the exception that "Penguins don’t fly". When we learn that Tweety is a bird, we conclude that Tweety flies. When we learn further that Tweety is a penguin, we don’t change our mind about the fact that most birds fly, penguins are birds that do not fly and Tweety is a bird, but we do change our mind about the fact that Tweety flies. This is a form of Non-monotonic reasoning, a way of deducing new information from given information in a manner that could possibly reduce the set of what is known, and hence does not satisfy the monotonicity property.

5 DEFEASIBLE REASONING

Defeasible reasoning is a form of non-monotonic reasoning. It aims to eliminate or modify the monotonicity property in systems, allowing for reasoning with exceptions, through the development of formalisms which can represent and reason with defeasible facts such as "Birds typically fly".[6] There are many frameworks which have been proposed for defeasible reasoning, and, unlike entailment for classical reasoning, it is commonly-accepted that defeasible entailment is not unique.[5]

5.1 The KLM Approach

The framework for defasible reasoning proposed by Kraus, Lehmann and Magidor (KLM)[20] is a well-known axiomatic approach, based on the concept of plausible inference. In the logic proposed by KLM, a plausible inference is represented by a defeasible implication operator of the form $\alpha \mid\sim \beta$ which is read as “typically, if α then β”. The idea is that $\alpha \mid\sim \beta$ means that α is a good enough reason to believe β.

5.1.1 *Other Defeasible Reasoning Approaches.*

There are many other approaches to defeasible reasoning. Default reasoning is based on concepts of plausible inference of the form “In the absence of any information to the contrary of α, assume α”.[25] Negation as failure is a reasoning approach that assumes $\neg\alpha$ holds from failure to derive α.[10] Autoepistemic Logic allows for the expression of knowledge and lack of knowledge about facts, as opposed to traditional logics which only allows the expression of facts.[9]. Other well-known defeasible reasoning approaches include include *Circumscription*[24] and *Probabilistic logic*[17].

5.1.2 *Motivation for using the KLM Approach.*

It has been shown that the semantics for automated reasoning methods of KLM[16], Default logic[26], Negation as failure[10], Autoepistemic Logic[9], Circumscription[3] and Probabilistic logic[17] approaches can all be characterized for Description logics. Hence it seems reasonable to believe that the semantics of any one of these approaches could be characterized for Datalog and DLV, which are both Description logics.

There are two main reasons, stated in [5], why the KLM approach to extending Description logics seems preferable. Since Defeasible entailment is not unique, it seems necessary to have a way of differentiating between acceptable and non-acceptable methods of entailment. The KLM framework provides a list of rationality properties, known as KLM properties. Lehmann and Magidor[22] argue that all KLM properties must be satisfied by defeasible entailment methods, and hence, these properties provide a mechanism for formal analysis of defeasible entailment methods in order to assess how rational, intuitive and acceptable they are. Furthermore the KLM approach allows for decision problems to be reduced to classical entailment checking when possible to avoid unnecessarily drastically increasing the computational complexity with respect to the underlying classical case.

5.2 KLM Extension to Propositional Logic

In [7] it was found that we can view the logic of KLM-style defeasible implications as an extension of propositional logic. In this section we will explore the syntax and semantics of this extension.

First we extend to syntax of propositional logic. Suppose the set of atoms $\mathcal{P}$, the set of formulas $\mathcal{L}$ and the set of all interpretations U are all defined by the usual definitions, presented in Section 2. The alphabet of propositional logic is now extended by adding $\mid\sim$ as an operator. The language of propositional logic is extended to include all possible formulas of the form $\alpha \mid\sim \beta$, where $\alpha, \beta \in \mathcal{L}$. Notice that the restriction of α and β to formulas of the propositional form prevents nesting of the defeasible implication operator.

Next, we consider extending the semantics of propositional logic. The semantics for propositional logic remain the same, with satisfaction of a formula $\alpha \in \mathcal{L}$ by an interpretation $v \in U$, denoted by $v \Vdash \alpha$, defined as in Section 2. Similarly, the models of a set of formulas X denoted $[\![X]\!]$ is defined as in Section 2.

We use structures called *ranked interpretations* to define the semantics of KLM-style rational defeasible implications, as defined in [7]. The idea is that a ranked interpretation is an organization of interpretations into levels (or ranks), in order of decreasing typicality, where no level is empty, starting at level 0, with the most typical interpretations, and ending at level n for some $n \in \mathbb{N}$, with the least typical interpretations. A final level, of infinite rank, is added for all the impossible interpretations. Formally, a *ranked interpretation* is a function $R : U \longrightarrow \mathbb{N} \cup \{\infty\}$ such that $R(u) = 0$ for some $u \in U$, and for every $i \in \mathbb{N}$, if $R(v) = i$, then, for every j s.t. $0 \leq j < i$, there is a $u \in U$ for which $R(u) = j$. We call $R(v)$ the *rank* of v with respect to R. We denote the set of possible interpretations, those with finite rank, by U^R. The *ranked model* of a formula $\alpha \in \mathcal{L}$, denoted $[\![\alpha]\!]^R$, is the set of possible interpretations $v \in U^R$ such that $v \Vdash \alpha$.

We say R *satisfies* classical statement α, denoted $R \Vdash \alpha$, if $U^R \subseteq [\![\alpha]\!]^R$ and we call R an *α-interpretation.* The idea of satisfaction for a defeasible implication is that a ranked interpretation R satisfies a defeasible conditional formula $\alpha \mid\sim \beta$ if all minimal α-interpretations with respect to rank (the α-interpretations in the first level which contains an α-interpretation) satisfy β in a propositional manner. It is shown in [7] that we can associate each ranked interpretation R with some total preorder $\preceq$ on U^R, with strict version $\prec$. Formally, we say R satisfies $\alpha \mathrel{|\!\sim} \beta$, denoted $R \Vdash \alpha \mathrel{|\!\sim} \beta$, if all the *possible* $\prec$-minimal α-interpretations also satisfy β. More generally, R satisfies a finite set of conditionals $\mathcal{K}$, called a *knowledge base*, if $R \Vdash \alpha \mathrel{|\!\sim} \beta$ for every $\alpha \mathrel{|\!\sim} \beta \in \mathcal{K}$.

5.3 The KLM Properties

We motivated for the use of the KLM approach mainly due to the list of *rationality properties* for formalizing what is meant by acceptable defeasible entailment, as proposed in [22]. As in [7], we will refer to the satisfaction of all these properties as *LM-rationality*. The KLM properties, or rationality properties, are listed below:

$$\text{(Ref) } \mathcal{K} \mathrel{|\!\approx} \alpha \mathrel{|\!\sim} \alpha \qquad \text{(LLE) } \frac{\alpha \equiv \beta,\ \mathcal{K} \mathrel{|\!\approx} \alpha \mathrel{|\!\sim} \gamma}{\mathcal{K} \mathrel{|\!\approx} \beta \mathrel{|\!\sim} \gamma}$$

$$\text{(RW) } \frac{\mathcal{K} \mathrel{|\!\approx} \alpha \mathrel{|\!\sim} \beta,\ \beta \models \gamma}{\mathcal{K} \mathrel{|\!\approx} \alpha \mathrel{|\!\sim} \gamma} \qquad \text{(And) } \frac{\mathcal{K} \mathrel{|\!\approx} \alpha \mathrel{|\!\sim} \beta,\ \mathcal{K} \mathrel{|\!\approx} \alpha \mathrel{|\!\sim} \gamma}{\mathcal{K} \mathrel{|\!\approx} \alpha \mathrel{|\!\sim} \beta \land \gamma}$$

$$\text{(Or) } \frac{\mathcal{K} \mathrel{|\!\approx} \alpha \mathrel{|\!\sim} \gamma,\ \mathcal{K} \mathrel{|\!\approx} \beta \mathrel{|\!\sim} \gamma}{\mathcal{K} \mathrel{|\!\approx} \alpha \lor \beta \mathrel{|\!\sim} \gamma} \qquad \text{(CM) } \frac{\mathcal{K} \mathrel{|\!\approx} \alpha \mathrel{|\!\sim} \beta,\ \mathcal{K} \mathrel{|\!\approx} \alpha \mathrel{|\!\sim} \gamma}{\mathcal{K} \mathrel{|\!\approx} \alpha \land \beta \mathrel{|\!\sim} \gamma}$$

$$\text{(RM) } \frac{\mathcal{K} \mathrel{|\!\approx} \alpha \mathrel{|\!\sim} \gamma,\ \mathcal{K} \not\mathrel{|\!\approx} \alpha \mathrel{|\!\sim} \neg\beta}{\mathcal{K} \mathrel{|\!\approx} \alpha \land \beta \mathrel{|\!\sim} \gamma}$$

5.4 Rational Closure

Rational Closure, proposed by Lehmann and Magidor[22], is the most conservative form of defeasible entailment (in terms of subset inclusion) which is LM-rational. However, according to [5], Rational Closure is usually too weak from an inferential point of view. As an example, Rational Closure does not support the inheritance of defeasible properties. Lexicographic Closure, a form of defeasible entailment proposed by Lehmann[21], is bolder in terms of subset inclusion and is still LM-rational. Other more expressive forms of defeasible entailment include Ranked Entailment[22] and Relevant Closure[5]. However, these are not LM-rational.[7] Both the appropriate forms of defeasible reasoning, Rational Closure and Lexicographic Closure, can be defined in the KLM propositional logic extension in [7] For simplicity sake, we will only consider Rational Closure. Furthermore, we motivate for the use of Rational Closure based on its relatively low computational complexity.

5.4.1 *The Idea and Formalization.*

The idea is that create a ordering of ranked interpretations, where the lower ranked interpretations are seen as more typical, on an *outer* level of typicality. On an *inner* level of typicality, within each ranked interpretation we know that lower interpretations are seen as more typical. So we want the minimum ranked interpretation, the most typical one (on an outer level), to be the one in which interpretations are as typical (on an inner level) as $\mathcal{K}$ allows them to be. A defeasible conditional formula $\alpha \mathrel{|\!\sim} \beta$ is in the rational closure of a knowledge base $\mathcal{K}$ if it is satisfied by the most typical ranked interpretation, and hence, "makes sense" typically.

We know, from Giordano et al. [16], that there is a unique $\preceq_{\mathcal{K}}$-minimal ranked interpretation, denoted $R^{RC}_{\mathcal{K}}$. We say $\alpha \mathrel{|\!\sim} \beta$ is in the rational closure of $\mathcal{K}$, denoted as $\mathcal{K} \mathrel{|\!\approx}_{RC} \alpha \mathrel{|\!\sim} \beta$, if $R^{RC}_{\mathcal{K}} \Vdash \alpha \mathrel{|\!\sim} \beta$. Lehmann and Magidor found that a defeasible entailment relation is LM-rational if and only if it can be generated from a ranked interpretation.[22] Hence, Rational Closure must be LM-rational.

5.4.2 *Algorithm for Base Ranks.*

We now consider how to go about finding the unique minimal ranked interpretation $R^{RC}_{\mathcal{K}}$, as defined in [7]. The *materializations* of $\mathcal{K}$ is $\overrightarrow{\mathcal{K}} = \{\alpha \to \beta : \alpha \mathrel{|\!\sim} \beta \in \mathcal{K}\}$. Intuitively, $\overrightarrow{\mathcal{K}}$ takes all the defeasible implications in $\mathcal{K}$ and turns them into classical implications.

We define an algorithm for ranking interpretations of a knowledge base $\mathcal{K}$ into *base ranks*, as done in [7]. First, we define a sequence of *materializations* $\mathcal{E}_0, \ldots, \mathcal{E}_{n-1}, \mathcal{E}_\infty$. We start with $\mathcal{E}_0 = \overrightarrow{\mathcal{K}}$. Then we inductively define each $\mathcal{E}_i = \{\alpha \to \beta \in \mathcal{E}_{i-1} : \mathcal{E}_{i-1} \models \neg\alpha\}$, for $i > 0$. Intuitively, we retain only the statements $\alpha \to \beta$ of $\mathcal{E}_{i-1}$, such that α can be proved false by all the statements not in $\mathcal{E}_{i-1}$. We stop this process when $\mathcal{E}_i = \mathcal{E}_{i-1}$, calling $n = i - 1$ and $\mathcal{E}_\infty = \mathcal{E}_n$.

Using the sequence $\mathcal{E}_0, \ldots, \mathcal{E}_{n-1}, \mathcal{E}_\infty$, we define the sequence $R_0, \ldots, R_{n-1}, R_\infty$ inductively, for $0 \leqslant i \leqslant n-1$, defining $R_i = \mathcal{E}_i \setminus \mathcal{E}_{i+1}$. Finally, we define $R_\infty = \mathcal{E}_\infty$. We say that R_i is the set of of classical implications $\alpha \to \beta$ (corresponding to defeasible implications $\alpha \mathrel{|\!\sim} \beta$) with *base ranks* i. Intuitively, if $\alpha \to \beta$ has *base rank* i then i is the smallest integer such that $\alpha \to \beta$ is true in at least one of the most typical interpretations in every ranked interpretation of $\mathcal{E}_i$.

5.4.3 *Algorithm for Defeasible Entailment.*

Finally, we consider how to go about checking for defeasible entailment, as defined in [7]. If we are checking for entailment of a classical statement α, denoted $\mathcal{K} \mathrel{|\!\approx} \alpha$, we check if α is entailed by all the statements in the infinite rank, the classical statements.

Otherwise, if we are checking for $\mathcal{K} \mathrel{|\!\approx} \alpha \mathrel{|\!\sim} \beta$, then we procede by removing the sets of classical implications $R_0, \ldots, R_{n-1}$ from $\overrightarrow{\mathcal{K}}$, in order, starting with the classical implications in R_0. We stop when we find the first $R = \overrightarrow{\mathcal{K}} \setminus R_0 \setminus \ldots \setminus R_m$, for some $m < n$, in which α is classically consistent. If it is the case that we get to the last level with $R_\infty \Vdash \alpha$ then $\mathcal{K} \mathrel{|\!\approx}_{RC} \alpha \mathrel{|\!\sim} \beta$ is true. When this is not the case, we check if R classically entails $\alpha \to \beta$, the materialisation of $\alpha \mathrel{|\!\sim} \beta$. If $R \Vdash \alpha \to \beta$ then $\mathcal{K} \mathrel{|\!\approx}_{RC} \alpha \mathrel{|\!\sim} \beta$ is true, otherwise it is false.

It can be proved that this algorithm returns true if and only if $\mathcal{K} \mathrel{|\!\approx}_{RC} \alpha \mathrel{|\!\sim} \beta$. [12] What is really desirable about this algorithm is that it consists only of a sequence of classical entailment checks, where the number of checks is linear with respect to the size of $\mathcal{K}$.[7] Hence, computing Rational Closure is only as hard as checking classical entailment.[7]

6 DATALOG

Logic programming is a programming paradigm based mainly on formal logic. There are many different types of logic programming with major logic programming language families such as *Prolog*, *Answer set programming(ASP)* and *Datalog*. We focus our attention on Datalog, a declarative logic programming language which is a syntactic subset of Prolog but with different semantics. We will first provide a motivation for the use of Datalog and then define both the syntax and semantics of Datalog.

6.1 Why is Datalog Useful?

Datalog is a popular query language for deductive databases, systems that can deduce new facts from a large amount of facts stored in a relational database based on a set of rules.[39] However, in recent year, many new applications of Datalog have become apparent, proving its usefulness and versatility. Datalog has been used for applications of data integration, declarative networking, program analysis, information extraction, network monitoring, security, and cloud computing.[18]

6.2 Syntax

Datalog is a simplified version of general Logic Programming. As such its syntax follows from that of a general logic programming language, and hence, from that of First-order logic. The language of Datalog is a set of formulas in the form of function-free Horn clauses. In this section we briefly, formalize this language and thus the syntax of Datalog, based on complete formalizations of Logic programming in [23][1] and Datalog in [8].

The alphabet, as in First-order logic, is made up of *variables* denoted by $X, Y, Z, \ldots$, *constants* denoted by $a, b, c, \ldots$ and *predicate symbols* denoted by $p, q, \ldots$. Since Datalog is function-free, a *term* is defined to be either a variable or a constant.

A *literal* has the form $p(t_1, \ldots, t_n)$, where p is a predicate and each t_i is a term called an *argument* to the predicate. The idea of a literal is equivalent to that of an atom as in First-order logic.

A *Horn clause*, or *definite clause*, which we will denote by $\alpha, \beta, \ldots$, has the form $h \leftarrow l_1 \wedge \cdots \wedge l_m$, where the left-hand side(LHS) h is a literal, called the *head* of the clause, and the right-hand side(RHS), called the *body* of the clause, is also made up only of literals l_i.[8] This corresponds to the idea of a formula (in first-order logic) $\forall X_1, \ldots, \forall X_m, (l_1 \wedge \cdots \wedge l_m \rightarrow h)$, where $X_1, \ldots, \forall X_m$ are all the variables occurring in the clause. We call the clause a *rule* if there is at least one literal on the RHS. Otherwise we call the clause a *fact*, which we denote by h for simplicity. Notice, this now corresponds to the idea of an atom h in first-order logic.

We call an *expression*, a term, literal or Horn clause, *ground* if it does not contain any variables. A *safety condition* is specified for Datalog, stating that every formula is either a ground fact or a rule such that every variable in the head of a clause also appears in the body of the clause. This safety condition ensures that only a finite number of facts can be derived from any Datalog program.[8]

6.2.1 Syntax and Relational Databases. In general logic programming all formulas are contained in a single logic program.[23] Datalog, however, is intended to be used by programs which use a large amount of facts stored in a relational database. So we need to consider two sets of clauses[8]:

- The *Extensional Database(EDB)* is a set of ground facts, which is physically stored in a relational database *DB*. We call the set of predicates occurring in *DB EDB-predicates.* Each EDB-predicate r corresponds to exactly one relation R in *DB*, such that each fact $r(t_1, \ldots, t_n)$ of *DB* is stored as a tuple $< t_1, \ldots, t_n >$ of R.
- The *Intensional Database(IDB)* is a Datalog program P. We call the set of predicates occurring in P but not in *DB IDB-predicates.* The head predicate of each clause in P is an IDB-predicate and EDB-predicates only occur in P in clause bodies.

6.3 Semantics

We will now briefly describe the semantics of the Datalog language in terms of Model Theory, as formalized in [8].

The *Herbrand Universe* U^P is the set of all constants in P. The *Herbrand Base* B^P is the set of all ground facts constructible from the symbols in P. We denote *EHB* as the set of all literals of B^P whose predicate is an EDB-predicate. Similarly, we denote *IHB* as the set of all literals of B^P whose predicate is an IDB-predicate.

As shown when defining the syntax of Datalog, we can associate each EDB-fact f and Datalog rule α with a first-order logic atom f^* and formula α^*, respectively. Then we can associate each set of Datalog clauses S with the conjunction S^* of all formulas α^* such that $\alpha \in S$. And, we define *cons*(S) to be the set of facts $f*$ such that $S^* \Vdash f^*$ in first-order fashion.

A *Herbrand interpretation* assigns each constant symbol to itself and each predicate symbol to a set of predicates ranging over constant symbols and is identified with a subset $\tau \subseteq B^P$. That is, Herbrand interpretations directly assign truth values to ground facts. Notice that this differs from normal semantics in First-order logic which is based of the concept of interpretations of constants, assigning truth values to constants.

A ground fact $p(t_1, \ldots, t_n)$ is true under τ if and only if $p(t_1, \ldots, t_n) \in \tau$. A Datalog rule $h \leftarrow l_1 \wedge \cdots \wedge l_m$ is true under τ if and only if for each substitution θ which replaces variables by constants, $l_1\theta \in \tau, l_2\theta \in \tau, \ldots, l_m\theta \in \tau$ implies that $h\theta \in \tau$. We say τ *satisfies* clause c if c is true under τ. If τ *satisfies* c then we call τ a *Herbrand model* for c. It is shown in [8] that *cons*(S) is the least Herbrand model of any set S of datalog clauses.

7 DLV

DLV (or DataLog with $\vee$) is a form of disjunctive logic programming. It is an extension of Datalog which allows disjunction ($\vee$) in the heads of rules and negation ($\wedge$) in the bodies of the rules. In this section we will first provide a motivation for the extension of Datalog and then define both the syntax and semantics of DLV.

7.1 Motivations for Extending DLV

DLV is considered to be a state-of-the-art implementation of DLP.[11] Disjunctive logic programming (DLP) is very expressive and, under widely-supported assumptions, it is strictly more expressive than disjunction-free logic programming.[11] This means that we can represent problems using DLP which we could not represent using disjunction-free logic programming. On the other hand, disjunction can also allow for representing problems in a simpler and more natural fashion than could be done using disjunction-free logic programming.[11] It is important to note, though, that the high expressiveness of disjunctive logic programming results in a higher computational complexity (in the worst case) for DLP programs.

Furthermore, in the context of deductive databases, DLP is recognized as a valuable tool for reasoning due to its ability to support natural modeling of incomplete knowledge.[1] It is clear that DLV and DLP have many important applications due to their increased expressivity and ability to handle incomplete knowledge. Thus, the extension of DLV to allow for defeasible reasoning should further increase its use cases by allowing for common-sense style reasoning.

7.2 Syntax

In Datalog, only rules and facts in the form of Horn clauses are allowed. For example a Datalog program could include the fact *Parent(p,c)* or the rule *GrandParent(g,c)←Parent(g,p)∧Parent(p,c)*. However in DLV we allow rules such as *Mom(p)∨Dad(p)←Parent(p)* or *Mom(p)←Parent(p)∧¬Dad(p)*.

The syntax of DLV, as defined in [11], is similar to that of Datalog, with the addition of a disjunction rule. A *disjunctive rule* is a formula $a_1 \vee \cdots \vee a_n \leftarrow b_1 \wedge \ldots \wedge b_k \wedge not b_{k+1} \wedge \ldots \wedge not b_m$, where $a_1, \ldots, a_n, b1, \ldots, b_m$ are literals, $n \geqslant 0, m \geqslant k \geqslant 0$, and *not* represents negation-as-failure. This, intuitively, means if we can derive $b_1, \ldots, b_k$ and we cannot derive $b_{k+1}, \ldots, b_m$ then we derive at least one of $a_1, \ldots, a_n$.

7.3 Semantics

There has been a lot of research into the semantics of DLP and many different semantics have been proposed.[11] The Answer Sets semantics, proposed by Gelfond and Lifschitz[14], as an extension of stable model semantics of disjunction-free logic programs[15], is a commonly accepted semantic for DLP. These semantics were chosen to be implemented by DLV. We will briefly explore the semantics of DLV as defined in [11].

The semantics of a *Herbrand Universe* U^P and a *Herbrand Base* B^P are as described in the semantics of Datalog. The *Ground Instantiation* of a rule r, denoted $Ground(r)$, is the set of rules obtained by applying all possible substitutions θ from variables in r to constants in U^P. More generally, $Ground(P) = \bigcup_{r \in P} Ground(r)$.

We first define the answer sets of a positive program P, a *not*-free program, using $Ground(P)$. An *interpretation* $\tau \subseteq B^P$ is a set of literals, as for Datalog. We say τ is called closed under positive program P if, for any $r \in Ground(P)$, a literal of the head of r is in τ whenever a literal of the body of r is in τ. We say τ is an *answer set* for positive program P if it is closed under P and minimal with respect to set inclusion. Notice, that an answer set is a special case of a Herbrand interpretation from datalog.

Now we give a reduction of general programs to positive ones and use it to define answer sets of general programs. The *Gelfond-Lifschitz transform* of a general ground program P with respect to a set $X \subseteq B^P$ is the positive ground program P^X, obtained by deleting rules $r \in P$ which contain $notl$ for some $l \in X$ and then deleting the $notb_i$ parts of the remaining rules. Then an *answer set* of P can be defined to be the set $X \subseteq B^P$ such that X is an answer set of $Ground(P)^X$.

Finally, we say P entails a ground literal l, denoted $P \Vdash l$ if l is satisfied by every answer set τ of P, where satisfaction is as for Herbrand interpretations in Datalog.

7.4 Extending Defeasible Reasoning to DLV

The main aim of this project is to extend DLV to inlcude defeasible reasoning capabilities. It has been shown, by Britz et al[4], that computing rational closure when using description logics is feasible. Furthermore, as in the propositional version of rational closure[7], checking defeasible entailment can be reduced to a sequence of classical entailment checks for description logics[4]. Since DLV can be seen as a less expressive form of description logic than that used by Britz et al,it seems reasonable to assume that we can define a way of computing rational closure for DLV.

As seen in the extension of KLM-style defeasible entailment to propositional logic[7], to extend KLM-style defeasible entailment to DLV, the first step would be to extend the syntax to include the defeasible implication operator, which we will denote by say $\mid\!\sim$. Similarly to the propositional extension, it seems reasonable that defeasible implications take the form $a_1 \vee \cdots \vee a_n \mid\!\sim b_1 \wedge \ldots \wedge b_k \wedge not b_{k+1} \wedge \ldots \wedge not b_m$, where $a_1, \ldots, a_n, b1, \ldots, b_m$ are literals from the DLV language as usual. As seen previously, this approach would prevent nesting of the defeasible implication operator. Hence, the approaches to checking entailment, as proposed for the propositional extension, should follow for the DLV case.

We noted, while defining the semantics of Datalog, that Herbrand interpretations assign truth values to ground facts directly. Furthermore, while defining the semantics of DLV, we noted that answer sets are just a special case of Herbrand interpretations. It seems that the way we interact with a ground fact in DLV intuitively corresponds to the way we interact with an atom in propositinal logic. Hence, the way we interact with an answer set in DLV intuitively corresponds to the way we interact with an interpretation in propositional logic. Thus, it seems reasonable to assume that concepts of entailment, using ranked interpretations, for the propositional extension, can be adjusted to use a form of ranked answer sets, where the idea of a ranked answer set would be that answer sets (Herbrand interpretations) are ranked based on their typicality.

8 CONCLUSIONS

Knowledge Representation and Reasoning is a well-established approach to artificial intelligence. Logics are used to represent knowledge and automated reasoning methods are used to reason about that knowledge. Propositional logic is easy to understand and define. However, it is not nearly expressive enough to reason in a manner that parallels that of human. Hence, it is often not useful for real-world applications. Many extensions to propositional logic exist, such as first-order logic and description logics. These provide increased expressivity but often with the trade-off of increased computational complexity for automated reasoning methods.

Classical reasoning is a form of reasoning that does not allow for uncertainty and exceptions to general rules. It is simpler to define and often has lower computational complexity than other approaches. However, reasoning with uncertainty, known as defeasible reasoning, closely parallels the reasoning methods used by humans. Thus, this form of reasoning has many useful applications in the real-world.

There are many approaches to defeasible reasoning. Even more, there are many forms of defeasible entailment. However, the approach introduced by Kraus, Lehmann and Magidor (KLM)[20] is favoured as it provides a set of KLM properties which can be used can be used to assess the correctness of any form of defeasible entailment. Lehmann and Magidor [21] provided a definition of the most conservative form of defeasible entailment, rational closure, and showed that it satisfied the the KLM properties.

Datalog is a declarative logic programming language which is used as a query language for deductive databases. It can be considered as a description logic. Datalog is seen to be a popular language with many current applications. However, due to the strict Horn clause structure and safety constraints, it has limited expressivity in certain modelling situations. DLV is an extension to Datalog, which loosens the rule structure, allowing for disjunction in rule heads and negation-as-failure (not) in rule bodies. As a result, it is found that DLV can represent problems which Datalog is not able to represent.

It has been shown that rational closure can be defined for propositional logic[7] and description logics[4], and that checking for defeasible entailment can be performed as a sequence of classical entailment checks in both cases. We conclude that the techniques presented by Casini et al[7] can be used to extend DLV to include defeasible reasoning.

REFERENCES

[1] Chitta Baral and Michael Gelfond. 1994. Logic programming and knowledge representation. *The Journal of Logic Programming* 19-20, SUPPL. 1 (may 1994), 73–148. https://linkinghub.elsevier.com/retrieve/pii/0743106694900256

[2] Mordechai Ben-Ari. 2012. *Mathematical Logic for Computer Science.* Springer London, London.

[3] Piero Bonatti, Carsten Lutz, and Frank Wolter. 2006. Description Logics with Circumscription. In *Proceedings of the Tenth International Conference on Principles of Knowledge Representation and Reasoning (KR'06).* AAAI Press, Lake District, UK, 400–410. http://dl.acm.org/citation.cfm?id=3029947.3030000

[4] Katarina Britz, Giovanni Casini, Thomas Meyer, Kody Moodley, Uli Sattler, and Ivan Varzinczak. 2017. *Rational Defeasible Reasoning for Description Logics.* Technical Report. University of Cape Town, South Africa. https://core.ac.uk/download/pdf/151756088.pdf

[5] Giovanni Casini, Thomas Meyer, Kodylan Moodley, and Riku Nortjé. 2014. Relevant Closure: A New Form of Defeasible Reasoning for Description Logics. In *Logics in Artificial Intelligence*, Eduardo Fermé and João Leite (Eds.). Springer International Publishing, Cham, 92–106.

[6] Giovanni Casini, T Meyer, Kody Moodley, and I Varzinczak. 2013. Towards Practical Defeasible Reasoning for Description Logics, In Proceedings of the 26th International Workshop on Description Logics. *CEUR Workshop Proceedings* 1014, 587–599.

[7] Giovanni Casini, Thomas Meyer, and Ivan Varzinczak. 2019. Taking Defeasible Entailment beyond Rational Closure. (2019), 18 pages.

[8] Stefano Ceri, Georg Gottlob, and Letizia Tanca. 1989. What you Always Wanted to Know About Datalog (And Never Dared to Ask). *Knowledge and Data Engineering, IEEE Transactions on* 1 (04 1989), 146 – 166. https://doi.org/10.1109/69.43410

[9] Francesco Donini, Daniele Nardi, and Riccardo Rosati. 1998. Autoepistemic Description Logics. *IJCAI International Joint Conference on Artificial Intelligence* 1 (05 1998).

[10] Francesco M. Donini, Daniele Nardi, and Riccardo Rosati. 2002. Description logics of minimal knowledge and negation as failure. *ACM Transactions on Computational Logic* 3, 2 (apr 2002), 177–225.

[11] Thomas Eiter, Wolfgang Faber, Christoph Koch, Nicola Leone, and Gerald Pfeifer. 2000. *DLV - A System for Declarative Problem Solving.* Technical Report. Institut fur Informationssysteme. 6 pages. http://arxiv.org/abs/cs/0003036

[12] Michael Freund. 1998. Preferential Reasoning in the perspective of Poole default logic. *Artificial Intelligence* 98 (1998), 209–235. http://www.sciencedirect.com/science/article/pii/S0004370297000532

[13] Jean H Gallier. 2003. Logic for Computer science: Foundations of automatic theorem proving. *Foundations of Physics* 1, June (2003), 534.

[14] Michael Gelfond and Vladimir Lifschitz. 1991. Classical negation in logic programs and disjunctive databases. *New Generation Computing* 9, 3 (01 Aug 1991), 365–385. https://doi.org/10.1007/BF03037169

[15] Michael Gelfond and Vladimir Lifschitz. 2000. The Stable Model Semantics For Logic Programming. *Logic Programming* 2 (12 2000).

[16] Laura Giordano, Valentina Gliozzi, Nicola Olivetti, and Gian Luca Pozzato. 2015. Semantic characterization of rational closure: From propositional logic to description logics. *Artificial Intelligence* 226 (05 2015). https://doi.org/10.1016/j.artint.2015.05.001

[17] Jochen Heinsohn. 1994. Probabilistic Description Logics. In *Uncertainty Proceedings 1994*, Ramon Lopez de Mantaras and David Poole (Eds.). Morgan Kaufmann, San Francisco (CA), 311 – 318. https://doi.org/10.1016/B978-1-55860-332-5.50044-4

[18] Shan Shan Huang, Todd Jeffrey Green, and Boon Thau Loo. 2011. Datalog and Emerging Applications: An Interactive Tutorial. In *Proceedings of the 2011 ACM SIGMOD International Conference on Management of Data (SIGMOD '11).* ACM, New York, NY, USA, 1213–1216. https://doi.org/10.1145/1989323.1989456

[19] Bjarni Jonnson and Alfred Tarski. 1952. Boolean Algebras with Operators. *American Journal of Mathematics* 74, 1 (jan 1952), 127.

[20] Sarit Kraus, Daniel Lehmann, and Menachem Magidor. 2002. Nonmonotonic Reasoning, Preferential Models and Cumulative Logics. *Artificial Intelligence* 44 (03 2002), 167–207. https://doi.org/10.1016/0004-3702(90)90101-5

[21] Daniel Lehmann. 1999. Another perspective on Default Reasoning. *Annals of Mathematics and Artificial Intelligence* 15 (11 1999). https://doi.org/10.1007/BF01535841

[22] Daniel Lehmann and Menachem Magidor. 1992. What does a conditional knowledge base entail? *Artificial Intelligence* 55, 1 (1992), 1–60.

[23] J. W. Lloyd. 1984. *Foundations of Logic Programming.* Springer-Verlag, Berlin, Heidelberg.

[24] John McCarthy. 1980. CircumscriptionâĂŤA form of non-monotonic reasoning. *Artificial Intelligence* 13, 1 (1980), 27 – 39. https://doi.org/10.1016/0004-3702(80)90011-9 Special Issue on Non-Monotonic Logic.

[25] Raymond Reiter. 1980. A Logic for Default Reasoning. *Artif. Intell.* 13 (04 1980), 81–132. https://doi.org/10.1016/0004-3702(80)90014-4

[26] Kunal Sengupta, Pascal Hitzler, and Krzysztof Janowicz. 2015. Revisiting Default Description Logics – and Their Role in Aligning Ontologies. In *Semantic Technology*, Thepchai Supnithi, Takahira Yamaguchi, Jeff Z. Pan, Vilas Wuwongse, and Marut Buranarach (Eds.). Springer International Publishing, Cham, 3–18.