Rational preferential reasoning for datalog *

Michael Harrison and Thomas Meyer

Centre for Artificial Intelligence Research, Department of Computer Science,
University of Cape Town
hrrmic014@myuct.ac.za and tmeyer@cs.uct.ac.za

Abstract. Datalog is a powerful language that can be used to represent explicit knowledge and compute inferences in knowledge bases. Datalog cannot represent or reason about contradictory rules, though. This is a limitation as contradictions are often present in domains that contain exceptions. In this paper, we extend datalog to represent contradictory and defeasible information. We define an approach to efficiently reason about contradictory information in datalog and show that it satisfies the KLM requirements for a rational consequence relation. Finally, we introduce an implementation of this approach in the form of a defeasible datalog reasoning tool and evaluate the performance of this tool.

Keywords: Datalog · Non-monotonic Reasoning · Preferential Reasoning · Defeasible Implication · Knowledge Representation · Rational Closure.

1 Introduction

Datalog [1] is a rule-based language that was originally designed as an effort to integrate efforts from the Artificial Intelligence and Database communities [7]. The aim of datalog was to provide a deductive database querying language that extended conjunctive queries with recursion [1], thereby allowing a relation (predicate) to be present in both the head and body of a rule. Datalog was derived from logic programming [15], with a key distinction being that datalog does not contain functions.

Datalog has been around since the eighties, but interest in it waned as there did not seem to be many compelling uses for it. Datalog has experienced some renewed interest in the past decade as the world moves towards greater levels of automation in most industries. Some of the areas where datalog is currently being used include data integration, declarative networking, program analysis, information extraction, network monitoring, security, and cloud computing [10]. Datalog has been used as the core of some very expressive and efficient Knowledge Representation and Reasoning systems, such as DLV [14] and RDFox [18]. DLV is a Disjunctive Logic Programming (DLP) system that uses an extended Disjunctive Datalog [8] as its kernel language. DLV is one of the most successful and widely used DLP engines.

* Partially funded by the CSIR-DST Inter-Bursary support programme

Most of these reasoning systems have limited applicability to real-world problems, though, as they can usually not reason about inconsistent or contradictory information. Contradictions often occur in domains that contain exceptions. Many systems cannot handle these contradictions due to the systems being monotonic [2]. Monotonicity is a usually desirable property of logical languages that states that previously concluded information cannot be revoked in light of new, contradictory information. The information concluded in a monotonic system can only ever be added to and never taken away. Systems capable of dealing with contradictions need to, therefore, be nonmonotonic. DLV is nonmonotonic, but only in the way that it can represent and reason about incomplete information. We desire a system that can model rules that will *generally* hold true but also permit *exceptions* without the user having to perform additional knowledge engineering.

We present a simple example of a case where we would want to be able to represent and reason about *exceptional* information:

Example 1. **Strict mammal knowledge base**

- All mammals don't lay eggs.
- All platypusses are mammals.
- All platypusses lay eggs.

If we wanted to query the knowledge base in *Example 1* to find out if platypusses lay eggs, traditional datalog reasoning systems (including DLV) would conclude that platypusses lay eggs and platypusses don't lay eggs, thereby returning an empty model (essentially saying that there can be no platypusses). This is obviously not a desirable result. We would like to extend datalog to represent *defeasible* knowledge. A defeasible rule is a non-strict rule that *typically* holds. A defeasible version of *Example 1* would be:

Example 2. **Defeasible mammal knowledge base**

- All mammals typically don't lay eggs.
- All platypusses are mammals.
- All platypusses typically lay eggs.

Defeasible rules, unlike strict rules, would not have to hold if they are contradicted by strict information.

A seminal approach to reasoning about defeasible knowledge is the *preferential* approach, as defined by Kraus, Lehmann, and Magidor [11, 13] (often referred to as the KLM approach). The KLM approach looks at nonmonotonic reasoning from a general and abstract point of view. The framework defines the requirements that the authors of the KLM approach believe a nonmonotonic inference procedure should meet in order to be considered a *rational* consequence relation. The KLM approach is defined in the propositional logic setting. The KLM approach has been successfully lifted to Description Logics [6] as a way to extend the Description Logic language ALC with nonmonotonic reasoning capabilities. A practical defeasible reasoning system [17] using the KLM approach

has even been implemented in the form of a plugin for *Protégé*, an ontology engineering program.

The KLM approach's abstract framework and impressive computational tractability make it an appealing candidate for application to the datalog setting. In this paper, we define a defeasible extension to datalog. We define an algorithm that performs a defeasible entailment check for our extended defeasible datalog language, as well as defining the supporting algorithms required to perform this kind of reasoning. We show that our approach meets the KLM requirements for a rational consequence relation. We then introduce an implemented system that can create, edit, and, critically, query a defeasible datalog program.

2 Language

We will first define vanilla datalog before we propose our extensions to it. A datalog program consists of a set of datalog rules. Datalog rules are expressions of the form

$$b_1, ..., b_n \rightarrow h_1$$

where $b_1, ..., b_n$ and h_1 are *literals*. The left-hand side of the rule is the *body* and the right-hand side of the rule is the *head*[1]. Traditionally, the head of a datalog rule contains only one literal but the body is made up of a conjunction of any finite number of literals. If all the literals in the body are true in a model of a datalog program, then the literal in the head is implied and added to the model.

In vanilla datalog, a literal is just an *atom* p. An atom is an expression $p(t_1, ..., t_m)$, where p is a *predicate* with arity m and $t_1, ..., t_m$ are *terms*. A term can either be a *constant* or a *variable*.

Datalog follows a model-theoretic semantics where the datalog program is viewed as a set of first-order sentences that describes the desired answer [1]. A datalog *interpretation* is an assignment of concrete meaning to all constant and predicate symbols in the datalog program [7]. A datalog fact is entailed ($\models$) by a datalog program if that fact is true under every model of the datalog program.

We define two language extensions to datalog:

1. A practical defeasible datalog language that is used in our implemented defeasible datalog reasoning.
2. A theoretical defeasible datalog language enriched with additional operators that is used purely to prove how our approach meets the KLM requirements for a *rational consequence relation* (defined in Section 4).

2.1 Practical defeasible datalog language

We need to define an extension to datalog that is capable of expressing defeasible knowledge.

[1] Datalog is often written with the head on the left and the body on the right, with a fullstop at the end of each rule. :– is also often used to represent the implication operator instead of →. This more traditional representation is used for the syntax of our implementation.

The first essential extension to the language is *negation*, $\neg$[2]. Any literal in the head or body can now be an atom p or a negated atom $\neg p$. Negation is necessary in order to be able to express any contradictory information.

Our next extension is *disjunction* $\vee$. We include disjunction as a nice-to-have, purely because our system is built on top of DLV, which uses disjunctive datalog as its underlying language. We, therefore, also only allow disjunction in the head between literals.

Our final extension is an essential one — we add a *defeasible implication* operator $\rightsquigarrow$[3]. This operator can be used in place of the traditional strict implication operator $\rightarrow$ when we want to express a defeasible rule.

2.2 Theoretical defeasible datalog language

We define further extensions to our defeasible datalog language, purely to express the postulates that characterise the language and in proving that the postulates hold for our approach.

We introduce disjunction in the body between literals and conjunction in the head between literals.

The defeasible datalog rule

$$b_1 \vee ... \vee b_n \rightarrow h1$$

corresponds to the logical sentence

$$\forall x_1, ..., x_m (b_1 \vee ... \vee b_n \rightarrow h1)$$

where $x_1, ..., x_m$ are all the terms occurring in the all the literals of the rule.

Similarly, the defeasible datalog rule

$$b_1 \rightarrow h1 \wedge ... \wedge h_n$$

corresponds to the logical sentence

$$\forall x_1, ..., x_m (b_1 \rightarrow h1 \wedge ... \wedge h_n)$$

where $x_1, ..., x_m$ are all the terms occurring in all the literals of the rule.

Furthermore, we introduce bottom $\bot$, which can be seen as shorthand for $p \wedge \neg p$.

Bodies of rules are seen as equivalent $\equiv$ if they are modelled by the same interpretations.

A defeasible datalog fact $\alpha \rightsquigarrow \beta$ is *defeasibly entailed* ($\mid\!\!\approx$) by a defeasible datalog program if, knowing α is true and based on the information contained in the defeasible datalog program, it would be sensible to (defeasibly) conclude that β is true.

If we have a set of defeasible rules $\mathcal{DR}$, then $\overline{\mathcal{DR}}$ is a set of strict versions $\alpha \rightarrow \beta$ of all defeasible rules $\alpha \rightsquigarrow \beta$ in $\mathcal{DR}$.

[2] $-$ is used to represent negation in our implementation.

[3] $:\sim$ is used to represent defeasible implication in our implementation.

3 A rational consequence relation for defeasible datalog

Kraus, Lehmann, and Magidor studied preferential consequence relations as an approach to nonmonotonic reasoning [11]. Lehmann and Magidor looked at a more restricted class of consequence relations, *rational* relations [12]. They argued that any reasonable nonmonotonic inference procedure should define a rational consequence relation. Rational relations are those that may be represented by a *ranked* preferential model. A ranked model is a preferential model for which there is a modular strict partial order [12].

A nonmonotonic inference procedure needs to meet properties known as the KLM postulates to be considered a rational consequence relation. The original KLM postulates were defined in propositional logic. We define defeasible datalog versions of the KLM postulates that characterise a rational consequence relation for ranked defeasible datalog models:

Reflexivity	$\mathcal{K} \mid\!\approx \beta \sim\!\!> \beta$
Left Logical Equivalence	$\frac{\beta \equiv \gamma,\ \mathcal{K} \mid\!\approx \beta \sim\!\!> \eta}{\mathcal{K} \mid\!\approx \gamma \sim\!\!> \eta}$
Right Weakening	$\frac{\mathcal{K} \mid\!\approx \beta \sim\!\!> \eta,\ \models \eta \rightarrow \gamma}{\mathcal{K} \mid\!\approx \beta \sim\!\!> \gamma}$
And	$\frac{\mathcal{K} \mid\!\approx \beta \sim\!\!> \gamma,\ \mathcal{K} \mid\!\approx \beta \sim\!\!> \eta}{\mathcal{K} \mid\!\approx \beta \sim\!\!> \gamma \wedge \eta}$
Or	$\frac{\mathcal{K} \mid\!\approx \beta \sim\!\!> \eta,\ \mathcal{K} \mid\!\approx \gamma \sim\!\!> \eta}{\mathcal{K} \mid\!\approx \beta \vee \gamma \sim\!\!> \eta}$
Cautious Monotonicity	$\frac{\mathcal{K} \mid\!\approx \beta \sim\!\!> \gamma,\ \mathcal{K} \mid\!\approx \beta \sim\!\!> \eta}{\mathcal{K} \mid\!\approx \beta \wedge \gamma \sim\!\!> \eta}$
Rational Monotonicity	$\frac{\mathcal{K} \mid\!\approx \beta \sim\!\!> \eta,\ \mathcal{K} \not\mid\!\approx \beta \sim\!\!> \neg \gamma}{\mathcal{K} \mid\!\approx \beta \wedge \gamma \sim\!\!> \eta}$

4 Rational preferential reasoning for defeasible datalog

In this section, we attempt to emulate *rational closure* (a specific construction given by KLM [11] that satisfies all the KLM postulates for a rational consequence relation) for our defeasible datalog. Our approach satisfies the KLM postulates — the full proofs are available online[4].

Our rational closure will rely on a *ranking* of the defeasible datalog rules in a defeasible datalog program. We need to construct a ranking of our defeasible datalog rules based on the *exceptionality* of each rule. If a rule is assigned a lower ranking, then that rule is more normal or general. If a rule is assigned a higher ranking, then that rule is more exceptional or specific. Only once we have this ranking can we perform rational closure to determine if a rule is defeasibly entailed by our knowledge base.

[4] `https://github.com/MindfulMichaelJames/proofs/blob/master/Proofs.pdf`

Firstly, we define an algorithm to determine which defeasible rules in a set of defeasible rules are exceptional with regards to that set of defeasible rules and an optional set of strict rules. A defeasible rule is exceptional with regards to a set of defeasible and strict rules if the body of the rule is not satifisfiable with regards to the set of defeasible and strict rules.

Platypusses would not be satisfiable in *Example 2*, meaning that the defeasible rule with platypusses as the body ("All platypusses typically lay eggs") will be exceptional with regards to the strict rule and the other defeasible rule in *Example 2*.

Algorithm 1: $exceptional(\mathcal{SR}, \mathcal{DR}_{\mathcal{E}})$

Input : A set of strict datalog rules $\mathcal{SR}$ and a set of defeasible datalog rules $\mathcal{DR}_{\mathcal{E}}$

Output: $\mathcal{DR}_{\mathcal{E}'} \subseteq \mathcal{DR}_{\mathcal{E}}$ such that $\mathcal{DR}_{\mathcal{E}'}$ is exceptional w.r.t. $\mathcal{SR}$ and $\mathcal{DR}_{\mathcal{E}}$

```
1 DR_E' := ∅;
2 foreach α ⇝ β ∈ DR_E do
3     if SR ∪ overline(DR_E) ⊨ α → ⊥ then
4         DR_E' := DR_E' ∪ {α ⇝ β};
5 return DR_E';
```

Next, we define an algorithm that computes the ranking of a set of defeasible rules. This algorithm starts by putting all the defeasible rules in rank 0. It then utilises **Algorithm 1** to determine which rules are exceptional to rank 0 and a set of strict rules. The exceptional rules are moved from rank 0 to rank 1. The same process is then performed to determine which rules in rank 1 are exceptional relative to rank 1 and the strict rules. Those exceptional rules will be moved to rank 2. This process is continued until either there are no more exceptional rules in a rank or all the rules in a rank are exceptional. If all the defeasible rules in a rank are exceptional, then the defeasible rules are strict rules represented as defeasible rules. These rules will be moved to the set of strict rules and represented as strict rules.

Going back to *Example 2*, the two defeasible rules ("All platypusses typically lay eggs" and "All mammals typically don't lay eggs") would start on rank 0. "All platypusses typically lay eggs" would be found to be exceptional with regards to the other defeasible rules in its current rank ("All mammals typically don't lay eggs") and the strict rules ("All platypusses are mammals"). "All platypusses typically lay eggs" would, therefore, be moved to rank 1.

Algorithm 2: *computeRanking*$(\mathcal{SR}, \mathcal{DR})$

Input : A knowledge base consisting of the set of strict datalog rules $\mathcal{SR}$ and the set of defeasible datalog rules $\mathcal{DR}$

Output: The ranking of defeasible rules $\mathcal{R}$ and set of strict rules $\mathcal{SR}$

$\mathcal{DR}_0 := \mathcal{DR}; \mathcal{DR}_1 := exceptional(\mathcal{SR}, \mathcal{DR}_0); n := 1; \mathcal{R} := \emptyset;$
while $\mathcal{DR}_{n-1} \neq \mathcal{DR}_n$ ***and*** $\mathcal{DR}_n \neq \emptyset$ **do**
  $n := n + 1; \mathcal{DR}_n := exceptional(\mathcal{SR}, \mathcal{DR}_{n-1});$
if $\mathcal{DR}_n \neq \emptyset$ **then**
  $\mathcal{SR} := \mathcal{SR} \cup \overline{\mathcal{DR}_n};$
for $i = 1$ *to* $n - 1$ **do**
  $R_{i-1} = \mathcal{DR}_{i-1} \setminus \mathcal{DR}_i;$
$R_i := \mathcal{DR}_i;$
return $\mathcal{R} = \bigcup_{j=0}^{j \leq i} R_j$ *and* $\mathcal{SR}$;

Once we have computed the ranking for a defeasible datalog knowledge base, we can go about checking if a defeasible rule is defeasibly entailed by the knowledge base. We do this by checking if the rule is in the rational closure of the knowledge base. Essentially, the rational closure algorithm looks at the portion of the knowledge base for which the queried rule is not exceptional (using **Algorithm 1**). The algorithm then checks if the rule classically follows from this portion of the knowledge base.

Going back to *Example 2* again, if we wanted to query that knowledge base to see if it defeasibly entailed the query "All platypusses typically lay eggs", we would first look at the portion of the knowledge base where the body of the query is satisfiable. Platypusses are not satisfiable when considering rank 0 ("All mammals typically don't lay eggs"), rank 1 ("All platypusses typically lay eggs") and the strict rules ("All platypusses are mammals"). When considering just rank 1 and the strict rules, though, platypusses are satisfiable. We will then look at the strict version of the remaining defeasible rules and the strict rules and see if a strict version of our query is classically entailed, which it is in this case.

Our approach can also check for defeasible entailment of strict rules. To do this, we check if the strict query is classically entailed by the strict portion of the knowledge base. This is worth mentioning, but it is not our focus so it shall not be discussed further here.

Algorithm 3: *RationalClosure*$(\mathcal{K}, \alpha \sim\!\!> \beta)$

Input : A knowledge base $\mathcal{K}$ that consists of the ranking $\mathcal{R}$ of the set of defeasible rules $\mathcal{DR}$ and the set of strict rules $\mathcal{SR}$, and a defeasible query $\alpha \sim\!\!> \beta$

Output: **True** iff $\alpha \sim\!\!> \beta$ is in the rational closure of the knowledge base consisting of defeasible rules $\mathcal{DR}$ and strict rules $\mathcal{SR}$, **False** otherwise

1 $i := 0; n :=$ number of ranks in $\mathcal{R} + 1$;
2 **while** $\bigcup_{j=i}^{j \leq n} \overrightarrow{\mathcal{R}_j} \cup \mathcal{SR} \models \alpha \rightarrow \bot$ *and* $i \leq n$ **do**
3 $\quad i := i + 1$;
4 **return** $\bigcup_{j=i}^{j \leq n} \overrightarrow{\mathcal{R}_j} \cup \mathcal{SR} \models \alpha \rightarrow \beta$;

5 A defeasible datalog querying system

We now introduce DDLV. DDLV is a system that performs prefential reasoning for defeasible datalog programs. The defeasible datalog language for defeasible datalog programs is defined in **Section 3.1** of this paper.

Defeasible datalog programs can be edited in DDLV for convenience, but its main feature is the capability to query if a defeasible datalog rule is entailed by a defeasible datalog program. DDLV achieves this by implementing the algortithms defined in **Section 4.1** of this paper.

DDLV creates a *ranking* of a defeasible datalog program using implementations of **Algorithm 1** and **Algorithm 2**. This ranking is computed when a defeasible datalog program is loaded into DDLV or edited in DDLV.

To check if a defeasible datalog rule is defeasibly entailed by a defeasible datalog program, DDLV utilises an implementation of **Algorithm 3**.

Algorithm 1 performs a classical datalog entailment check on line 3. **Algorithm 3** performs two classical datalog entailment checks — one on line 2 and one on line 4. DDLV uses DLV [14] to perform these classical datalog entailment checks in co-NP. One of the greatest appeals to the KLM approach is its computational tractability and that is largely because the preferential reasoning process can be reduced to classical reasoning. This feature allows us to leverage the years of development that has been put into DLV to make it a highly efficient datalog reasoning system.

DDLV is built in Java and uses the DLV Wrapper [19] to interact with DLV programatically. The source code for DDLV is freely available online[5], along with instructions on how to install, run, and use DDLV.

6 Evaluation

This section evaluates the performance of DDLV. Seeing as though there are no other preferential reasoning systems for defeasible datalog, we compare DDLV

[5] `https://github.com/MindfulMichaelJames/DDLV`

with DIP, the Defeasible-Infererence Platform for Description Logics [16]. Although datalog and Description Logics have their differences, everything else about DIP is quite similar to DDLV. DIP performs KLM-style rational closure for ALC. The exceptionality check for DIP terminates in EXPTIME. Moodley evaluated DIP as part of his PhD thesis[17]. He synthesised ontologies to evaluate. Seeing as we have defined defeasible datalog, no defeasible datalog programs exist, so we must synthesise defeasible datalog programs to evaluate. We follow Moodley's parameters and techniques for the synthesised programs in order to have defeasible datalog programs that are as comparable as possible to his defeasible ontolgies. Hardware with identical specifications is also used (Intel i7, 4 cores, 3GB RAM).

We evaluate 10 groups of defeasible datalog programs. Each group has a different percentage of defeasible rules in the program, starting from 10% and going up to 100% in intervals of 10%. Each group contains 35 programs, with the smallest program containing 150 rules and the largest program containing 3500 rules. The program sizes are uniformly distributed between the smallest and largest programs.

Computing the ranking is the greatest bottleneck with this approach. Once the ranking has been computed, defeasible entailment checks can be performed very quickly. We will first compare DDLV's ranking compilation time with DIP's.

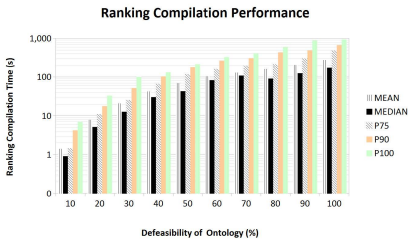

Fig. 1: *DIP ranking compilation time*

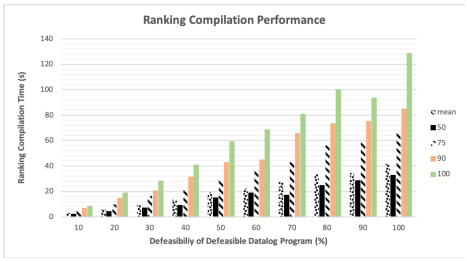

Fig. 2: *DDLV ranking compilation time*

Moodley used percentile plots because they give a good general picture of the performance and reveal outliers quickly [17]. Note that the vertical scale for *Figure 1* is staggered whereas the vertical scale for *Figure 2* is constant.

As with DIP, the time that DDLV takes to compute rankings increases with the level of defeasibility present in the program. DDLV seems to perform favourably against DIP, though. DIP already takes about 100 seconds on average to compute a ranking for an ontology with 60% defeasibility. DIP, in the worst case of datalog programs with 100% defeasibility, only takes just over 40 seconds on average to compute a ranking. The longest DIP takes to compute a ranking is nearly 1000 seconds. The longest DDLV takes to compute a ranking is less than 130 seconds.

Next, we will compare the time DDLV takes to perform a defeasible entailment check with the time DIP takes to perform a defeasible entailment check.

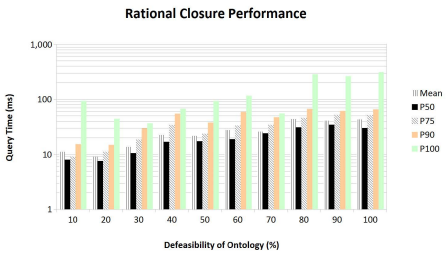

Fig. 3: *DIP rational closure performance*

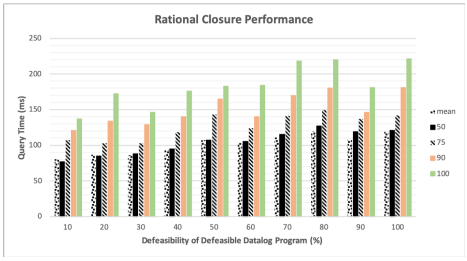

Fig. 4: *DDLV rational closure performance*

DDLV does not outperform DIP on average when it comes to defeasible entailment checks. DIP's average query time is less than 50 milliseconds, even for ontologies with 100% defeasibility. DDLV's average query time is about 120 milliseconds in the worst categories. In the worst cases, however, DDLV outperforms DIP. The longest DIP takes to perform a defeasible entailment check is just over 300 millisconds whereas the longest DDLV takes to perform a defeasible entailment check is about 220 milliseconds.

Even though DDLV's average query time is slower than DIP, the difference is very small in terms of real units of time and almost negligible when single queries are performed in isolation.

DDLV's impressive ranking time is to be celebrated, because getting the bottleneck of the ranking time to be as short as possible will greatly increase the usability of the system.

To test how well DDLV will scale up, we also present stress tests.

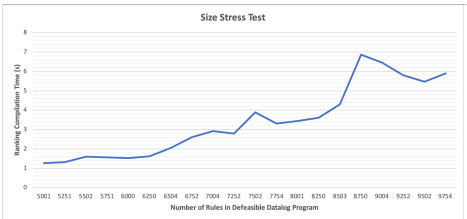

Fig. 5: *DDLV size stress test*

Fig. 6: *DDLV number of ranks stress test*

For the size stress test, we recorded the ranking compilation time for 20 defeasible datalog programs varying uniformly in size from 5000 rules to 10000 rules, with all 20 programs having a 20% level of defeasibility. For the ranking depth stress test, we recorded the ranking compilation time for 20 defeasible

datalog programs of 5000 rules and a 20% level of defeasibility, while the number of ranks in each program varied from 2 to 21. These two tests were performed because the number of rules in a program and the number or defeasible ranks have shown to have the greatest impact on ranking time. The two stress tests were performed on an Intel Xeon processor with 96 cores. DDLV performs the exceptionality checks for each rule in a rank asynchronously. This allows DDLV to perform as many exceptionality checks in parallel as the number of cores available. We believe this approach is the reason for the impressive results shown in *Figure 5* and *Figure 6*. The longest DDLV took to compute a ranking in the size stress test is just under 7 seconds for 8750 rules. The longest DDLV took to compute a ranking in the ranking depth stress test is just under 6.5 seconds for 19 ranks.

7 Related work

The KLM preferential reasoning approach has been lifted to the Description Logic setting a couple of times in a couple of different flavours [3][4][5][6]. DIP is the most similar work and the only other known implementation of KLM-style preferential reasoning.

There does not seem to be much work on handling inconsistent datalog programs. There is research into handling incomplete knowledge in datalog programs [9], but that is not the same type of nonmonotonicity that we are dealing with.

Morris et al., in a submission to this conference, consider the extension of a version of defeasible datalog to relevant closure and lexicographic closure. Their considerations are purely theoretical.

8 Conclusions and future work

We have identified the need for an extension to datalog that can handle inconsistent information in order to deal with exceptions. We introduced the defeasible datalog language that is able to express defeasible datalog rules and contradictory datalog rules.

We lifted the KLM preferential reasoning framework to our defeasible datalog setting. We defined versions of the rational closure algorithm and its supporting ranking and exceptionality algorithms for defeasible datalog. We defined defeasible datalog versions of the KLM postulates and proved that our rational closure algorithm meets these KLM requirements to be considered a rational consequence relation.

Finally, we introduced an implementation of our defeasible datalog reasoning approach that can perform efficient defeasible entailment checks for defeasible datalog programs.

In this paper, our preferential reasoning approach was very syntactic. Future work would involve clearly defining a semantics for this work and demonstrating the correspondence between the syntactic and semantic approaches.

References

1. Abiteboul, S., Hull, R., Vianu, V.: Foundations of Databases. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1st edn. (1995)
2. Ben-Ari, M.: Mathematical logic for computer science. Springer Science & Business Media (2012)
3. Britz, K., Heidema, J., Labuschagne, W.: Semantics for dual preferential entailment. Journal of Philosophical Logic **38**(4), 433–446 (2009)
4. Britz, K., Heidema, J., Meyer, T.: Semantic preferential subsumption. In: Eleventh International Conference on Principles of Knowledge Representation and Reasoning. pp. 476–484 (2008)
5. Britz, K., Meyer, T., Varzinczak, I.: Semantic foundation for preferential description logics. In: Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics). vol. 7106 LNAI, pp. 491–500 (2011)
6. Casini, G., Straccia, U.: Rational Closure for Defeasible Description Logics. In: European Workshop on Logics in Artificial Intelligence. pp. 77–90. Springer (2010)
7. Ceri, S., Gottlob, G., Tanca, L.: What You Always Wanted to Know About Datalog (And Never Dared to Ask). IEEE Transactions on Knowledge and Data Engineering **1**(1), 146–166 (1989)
8. Eiter, T., Gottlob, G.: Disjunctive Datalog. ACM Transactions on Database Systems **22**(3), 364–418 (1997)
9. Eiter, T., Leone, N., Mateis, C., Pfeifer, G., Scarcello, F.: A deductive system for non-monotonic reasoning. In: International Conference on Logic Programming and Nonmonotonic Reasoning. pp. 363–374. Springer (1997)
10. Huang, S.S., Green, T.J., Loo, B.T.: Datalog and emerging applications. Proceedings of the 2011 international conference on Management of data - SIGMOD '11 p. 1213 (2011)
11. Kraus, S., Lehmann, D., Magidor, M.: Nonmonotonic reasoning, preferential models and cumulative logics. Artificial Intelligence **44**(1-2), 167–207 (1990)
12. Lehmann, D.: Another perspective on default reasoning. Annals of Mathematics and Artificial Intelligence **15**(1), 61–82 (1995)
13. Lehmann, D., Magidor, M.: What does a conditional knowledge base entail? Artificial Intelligence **55**(1), 1–60 (1992)
14. Leone, N., Pfeifer, G., Faber, W., Eiter, T., Gottlob, G., Perri, S., Scarcello, F.: The DLV System for Knowledge Representation and Reasoning. ACM Transactions on Computational Logic (TOCL) **7**(3), 499–562 (2002)
15. Lloyd, J.W.: Foundations of logic programming. Springer Science & Business Media (2012)
16. Meyer, T., Moodley, K., Sattler, U.: DIP: A defeasible-inference platform for OWL ontologies. In: CEUR Workshop Proceedings (2014)
17. Moodley, K.: Practical Reasoning for Defeasible Description Logics. Ph.D. thesis, University of KwaZulu-Natal (2015)
18. Nenov, Y., Piro, R., Motik, B., Horrocks, I., Wu, Z., Banerjee, J.: Rdfox: A highly-scalable rdf store. In: International Semantic Web Conference. pp. 3–20. Springer (2015)
19. Ricca, F.: A Java wrapper for DLV. CEUR Workshop Proceedings **78**, 305–316 (2003)