Implementing Defeasibile Datalog in RDFox

Joshua J. Abraham[*]
University of Cape Town
Department of Computer Science
joshuasl8a@gmail.com

Thomas Meyer[†]
University of Cape Town
Department of Computer Science
tmeyer@cs.uct.ac.za

KEYWORDS

Propositional logic, Datalog, Defeasibility

Abstract

This literary review will centre around how one might implement the concept of nonmonotonicity and defeasibility into RDFox - a classical reasoner, written in C++, that is based on datalog - through the use of a defined algorithm that was designed to handle defeasibility in databases. A high level background will be given concerning the relevant theory, as well as the tools to be used (i.e. datalog and RDFox). This will then be followed by a discussion about and explanation of the algorithm to be incorporated. Finally, there will be a discussion on similar work done in this field and on this particular topic. This review will not focus as much on the theoretical side of the implementation, as it will the computation and practical side of it - that is, the implementing of defeasibility in RDFox. However, if one requires more information on the theory, one could find it in this works partner lierature review, (Pownall and Meyer, 2018) [1].

1 INTRODUCTION

Logic and reasoning have been conceptualised since before the days of Plato, through to Aristotle, medieval thinkers and is still a topic of discussion amongst present day philosophers, mathematicians, scientists, etc. There are many varying forms of logic and reasoning. For human beings it is relatively common and quite natural to adjust our views and beliefs based on new information. Even when faced with new information that is seemingly contradictory to what we already know, the logical mind has a natural ability and tendency to rationalise this new information and either add it to our base of knowledge or reject it entirely. However, when trying to simulate the efficiency and effectiveness of this same reasoning in a program or machine, the task becomes considerably more challenging. Imagine, if you will, being told that all birds fly and also that some animal, called Tweety, is a bird. From this base of information, it is reasonable to deduce that Tweety can fly. However, if it is then learned that Tweety is in fact a penguin or even an ostrich, would raise clear contradictions (as penguins and ostriches are examples of flightless birds). The type of reasoning called upon to handle situations such as this one is known as *defeasibile reasoning* or non-monotonic reasoning and is the type of reasoning this paper wishes to explore.

Defeasibile reasoning is a form of *default reasoning* and is employed when faced with seemingly contradictory information, opposed to what is already known or has already been reasonably deduced. This type of reasoning is often used in law, where contracts can be annulled in light of new evidence; in medicine, where medical diagnoses can be reevaluated due to the revelation of new symptoms; in science, where scientific theories can be falsified by ascertaining new experimental results; etc. Thus, it is clear to see the implicational benefits that would emerge from granting machines the power to employ this type of reasoning - this is one of the motivations behind this paper's work.

The goal of this paper is to explore a way to extend the expressivity of classical datalog - by means of creating a wrapper for the datalog reasoner, RDFox - with the form of non-classical reasoning, known as defeasible reasoning, that was just introduced. Many have already endeavoured to implement the concept of defeasibility into description logics (DL), however, this has never been done for RDFox before. The project, in it's entirety, involves a theoretical component - defeasible reasoning is incorporated into classical datalog and it's completeness and soundness are proven - and an implementational component - which is the focus of this paper in particular.

2 BACKGROUND

In this section a brief description of the key topics/concepts, which are indispensable in this research, are given. These are namely: *classical reasoning*, *propositional logic*, *defeasible reasoning*. Along with this, the tools to be used in this research project will be discussed. These are namely: *datalog* and *RDFox* (the RDF datalog reasoner). The descriptions and discussions follow now.

2.1 Classical Reasoning

Classical reasoning (also *classical first-order logic*) is the standard form of reasoning or drawing conclusions from already known propositions and/or *predicates*. There are three main laws that must stand when discussing any form of classical reasoning. These three laws are listed below, followed by a description of each:

(1) The law of identity
(2) The law of non-contradiction
(3) The law of excluded middle

The law of identity states that a thing is itself and can not be anything else. In terms of *propositional functions*, if there is some propositional function F for which F is true for some variable x, then F is indeed true for some variable x and can not be false or undefined in the same instance.

[*]Joshua is the student doing this particular research as his year project for Hons..
[†]Thomas Meyer is the research supervisor and student mentor.

The law of non-contradiction states that there exists no proposition A for which both A and $\neg A$ are true, i.e. $\neg(A \land \neg A)$ must always hold true.

The law of excluded middle states that, for every proposition, either the proposition itself is true and it's negation is false or the proposition is holds false and it's negation is true - i.e. there is no third truth value a proposition can hold. This law of excluded middle can be represented in propositional syntax form as $A \lor \neg A$.

The pitfalls of reasoning in a solely classical sense is that one cannot overcome the issue of defeasibility. These limitations explicitly arise in Chisholm's paradox of contrary-to-duty imperatives [2][3], as well as within the domains of law [4]. Furthermore, these limitations will be discussed in more depth in the next section of this paper.

2.2 Propositional logic

Propositional logic (also *sentential/statement logic*) is based off of classical reasoning and therefore adheres to the ideas and laws surrounding classical first-order logic. Simply put, a proposition is a sentence or statement that is either true or false and never both in the same instance. A proposition that is true has a truth value of true and similarly one that is false has a truth value of false. Propositional logic is the type of logic that deals with the combination or modification of propositions, as a whole, in order to generate more complex propositions - whose truth values generally reveal different or more detailed information about the related system or 'world' in which they exist. The simplest or fundamental propositions are those propositions that are indivisible, i.e. whose parts are not propositions themselves. Atomic propositions (atoms) are those propositions whose truth values do not depend on the truth values of other propositions - note that an atomic proposition is not necessarily a fundamental proposition, but a fundamental proposition can be an atomic proposition. A molecular proposition are those propositions that are generated by the joining of two or more propositions by means of *connectives* (logical operators). The five main connectives used in propositional logic, listed in order of precedence, are:

- $\neg$ (negation operator or logical 'not')
- $\land$ (conjunctive operator or logical 'and')
- $\lor$ (disjunctive operator or logical inclusive 'or')
- $\longrightarrow$ (implication or logical 'if, then')
- $\leftrightarrow$ (biconditional operator or logical 'if and only if')

If N and M be are propositional atoms, then a description of these logical connectives can be given by means of the following truth table:

N	M	¬ N	N ∧ M	N ∨ M	N → M	N ⟺ M
True	True	False	True	True	True	True
True	False	False	False	True	False	False
False	True	True	False	True	True	False
False	False	True	False	False	True	True

An example of a complex molecular proposition, that combines all the presented logical operators, would be:

N	M	¬ (N ∧ M ∨ (M → N)) ↔ N
True	True	False
True	False	False
False	True	False
False	False	True

Additionally, the Tweety-bird example, which was mentioned previously, can also be expressed in terms of propositional logic statements. Let B, T, P and F be propositional atoms representing bird, Tweety, penguin and flight respectively. Then it follows that, $B \longrightarrow F$, $T \longrightarrow B$, are properly formed propositional statements. The contradictions, that come from the logical *inference* of $P \longrightarrow F$, will be addressed when discussing the limitations of this particular logic.

Let it also be noted that this given set of logical connectives can also be extended to include $\oplus$ (exclusive 'or'), $\downarrow$ (negative inclusive 'or'), $\uparrow$ (negative 'and'), etc. However, the five given operators can be used to represent any extended operators by means of equivalent molecular statements. Similarly, it is also possible to reduce the set of connectives to a single connective. That is, to represent all the given connectives by means of equivalent molecular propositions that consist of only one connective - the Sheffer's stroke [5]. However, for the purposes of this paper, the middle ground will be taken and given set of connectives will be made use of, as it is rich enough to easily understand, yet sparse enough to keep the logical language simple.

The complete set of truth values of molecular propositions also provides information concerning the specific proposition in question. A tautology is a molecular proposition whose truth values is true in every instance, i.e. for all truth value couplings of the fundamental propositions it comprises of. Conversely, contradictions are those molecular propositions whose truth values are false in every instance. Contingencies are those molecular statements that are neither tautologies nor contradictions, i.e. they have at least one true truth value as well as at least one false truth value. Lastly, if the biconditional statement N $\iff$ M is a tautology, then the statements N and M are logically equivalent.

The inference (*entailment*) of propositional statements stems from the idea of classical reasoning, that is to deduce implicit propositions from explicitly stated ones. In the Tweety-bird example, $P \longrightarrow F$, is an example of inference. The rules for inference won't be discussed in this paper, but are stated in [6] [9] [10]. Similarly, an algorithm to deduce propositional satisfiability is presented as the Davis-Putnam algorithm [7] [8].

The limitations of propositional logic are now discussed - there are two main limitations that will be discussed in this paper. The first is the fact that, due to the nature of it, propositional logic does not allow for the use of variables in propositional statements (propositional formulae). For this, an extension of propositional logic, called *predicate* logic, has to be made use of. Note that predicate logic, like propositional logic, is also based on classical reasoning. In predicate logic, a *triple* is analogous to a proposition, which is of the form <subject - *predicate* - object>, and generally describes relations of the form 'is-a'. As an example the propositions, $B \longrightarrow F$

and $T \longrightarrow B$, can be respectively written as the triples, <B - can - F> and <T - is - B>. In a similar manner a propositional formula can be constructed and is of the form <x - is - B> ∧ <x - is - P> → <x - cannot - F>, where x is the predicate variable that can be substituted for any propositional atom, e.g. T. With this example it is clear to see the contradiction that arises and serves as a lead into the second limitation.

The second limitation stems from the fact that propositional logic is based on classical reasoning and is simply the fact that propositional logic does not cater for defeasibile propositions or typicality. Note that predicate knowledge has the same limitation as well. From the previous example it is clear to see that if x is a bird then x can fly, however, if x is a penguin as well then x cannot fly. This contradiction is a breach of classical reasoning's law of non-contradiction and therefore means that, either the proposition that birds fly or the one that penguins don't, has to be dropped in order to main soundness and completeness. However, it is intuitively known that neither of these statements should be dropped as they are both, in their own right, true. Thus, a new form of reasoning needs to be introduced to cater for these cases of typicality. This reasoning is defeasibile reasoning and is this paper's next topic of discussion.

2.3 Defeasible Reasoning

Defeasibile reasoning is a form of *default reasoning* and is employed when faced with seemingly contradictory information, opposed to what is already known or has already been reasonably deduced. Defeasibile reasoning can be viewed as an extension of propositional logic and allows the ability to follow complex patterns and deductions which, as shown, is not possible to be followed by means of classical reasoning alone. This is done by introducing the concept of typicality. The defeasible consequence logical operator, $\rightsquigarrow$, is used to represent typicality and is syntactically analogous to the classical implication operator, $\longrightarrow$. With this, we are finally able to solve the apparent inconsistencies in the Tweety-bird problem. Instead of saying that '*all birds fly*', consider the phrase: *typically birds have the ability of flight*. This entails that Tweety - which was initially classified to have the ability of fly due to the fact that it is a bird - can now be reclassified to not have the ability of flight, upon learning that Tweety is a penguin, and still maintain that Tweety is a bird. In defeasibile propositional logic syntax, this is expressed as,

$birds \rightsquigarrow flight$,

which should be read as '*birds typically fly*'.

Note that defeasibile logic is not the only approach to *default reasoning*. *Logical Programming without Negation as Failure* is one other approach. However, defeasibile logic outperforms the latter in both performance and expressivity. The results of the comparison of these two default reasoning approaches are explained, in more depth and detail than this paper is able to allow, in (Antoniou, Maher and Billington, 1999) [11].

2.4 Datalog

The declarative logic programming language, *datalog*, is most often used as a query language for deductive databases. It is based upon predicate logic and is thus sound, complete and, more importantly for the purposes of this paper, is also a well established model for propositional logic. Datalog, however, is not *Turing complete* and is thus mainly used in the areas of logic programming; knowledge representation and database querying, although it has recently been found useful in a number of other domains such as data integration and data extraction. Advantageously, this makes it easier to make full use of efficient algorithms developed for query resolution, which is one of the motivations behind choosing datalog for the purposes of this paper. Furthermore, the effectiveness of datalog can be seen by considering the many widely used database systems which incorporate ideas and algorithms developed specifically for datalog.

Furthermore, datalog is a syntactic subset of it's precursor, prolog.

2.5 RDFox

Since there is an unthinkably large amount of data flowing through the World Wide Web at any given time, it is a proportionally large benefit to have a structured form of data describing this data (metadata) to improve the discovery of and access to the data it's describing. However, this metadata requires a defined syntax and structure, in order to be machine-readable. The Resource Description Framework (RDF) was developed in collaboration by members of the World Wide Web Consortium [12], as a solution to this issue. The article in [15] gives a thorough description on what exactly RDF is and how it works. For the purposes of this paper all that needs to be known, concerning RDF, is that a RDF store (triple store) is simply a database built with the intent of storing and retrieving triples through semantic queries.

RDFox is an RDF store developed at the University of Oxford [13] and can be used as a datalog reasoner. RDFox is centralised (i.e. RDF data is stored on main memory) and allows for large growth within the RDF knowledge representation store. Triples can be added to a RDFox database by means of importing them or scheduling them for incremental addition or deletion. Datalog rules are added in the same manner. This method of adding triples and rules to a database allows RDFox to compute parallel datalog reasoning by means of incremental materialisation. This simply means that when the triple,
<Tweety - is - bird>,
is explicitly stored in the database, along with the rule,
bird $\implies$ flight,
the triple,
<Tweety - can - fly>,
is implicitly generated (materialised) as well. This check for inference is done incrementally, for each new triple and/or rule that enters the database. RDFox also supports SPARQL query answering [14].

3 EXPLANATION AND IMPLEMENTATION OF THE DEFEASIBILITY ALGORITHM

In this section the algorithm, which will be used to implement defeasibility into RDFox, will be discussed and explained. This algorithm is based on the one originally presented in (Casini and Stracia, 2010) [22]. This algorithm consists of three successive functions; their descriptions are as follows:

(1) *Exceptionality*: determining exceptions within a knowledge base
(2) *Ranking*: Computing an ordered ranking of exceptions based on their strength (i.e. how 'concrete' they are)
(3) *Querying*: Dealing with queries that posses a defeasibile element

These functions are still to be defined and explained in further detail in this section. Furthermore, since this paper focuses on the implementation of this algorithm in RDFox, pseudo code of each function will also be provided. The provided pseudo code and explanations of these three functions are all reiterated summaries of their original presentation in (reference Introducing Defeasibility into OWL Ontologies, Casini and Meyer etc.)

First, the term *conditional knowledge base* needs to be defined. A conditional knowledge base, K, is of the form K = <T, D>. Here T is the set of definitions and specialisations (i.e. concrete knowledge or classical formulae), also known as a TBox, and D is the set of defeasible inclusions (i.e. typical consequences or defeasibile formulae), also known as a *defeasible* TBox. With respect to the penguin example, these are expressed as T = {penguin $\longrightarrow$ bird, penguin $\longrightarrow$ $\neg$ fly} and D = {birds $\rightsquigarrow$ fly}. Furthermore, the computation of the materialisations, $\overline{D}$, of the inclusions in D - which are of the form $A \rightsquigarrow B$, where A and B are propositional atoms - is simply explained in set theory notation by $\overline{D} = \{\neg A \sqcup B \mid A \rightsquigarrow B \in \mathrm{D}\}$ (Casini and Stracia, 2010) [22].

The first function, *exceptionality*, can now be explained. A propositional atom, A, is exceptional with respect to some knowledge base K = <T, D> if and only if T = $\models \sqcap \overline{D} \longrightarrow \neg$ A. Thus, some defeasibile inclusion, $A \rightsquigarrow B \in$ D, is exceptional with respect to K if its antecedent is exceptional (i.e. A). The *exceptionality* function takes in T and some D' $\in$ D of K and produces a subset of D', E, which contains all the exceptions in D'. The pseudo code is provided as follows:

Algorithm 1 *Exceptional* (T, D')

```
1: let E = []
2: for $A \rightsquigarrow B$ in D' do
3:     if T $\models \bigsqcap \overline{D'} \longrightarrow \neg$ A then
4:         extend E by $[A \rightsquigarrow B]$
   return E
```

The second function, *ranking*, is now going to be explained. The *ranking* function deals with exceptions within exceptions by assigning each exception a rank based on the level of exception they are. This is done by computing *exceptional* and then repeating it's execution with it's previous iteration's output, until it's previous and current output are equivalent sets (i.e. further iterations will produce no change). The *ranking* function takes in the elements of some knowledge base, K = <T, D>, and outputs the knowledge base, K* = <T*, D*> and R = D_0, ..., D_n. Here, T* contains all the elements of T, as well as the irrefutable implicit rules in D; D* contains all the implicit rules in D that are not also in T*, which are also ranked in regards to their exceptionality; R is the partitioned set of D and each D_i is the set of defeasible rules with ranking i. The *ranking* algorithm in pseudo code is as follows:

Algorithm 2 *Ranking* (T, D)

```
1: let T* = T, D* = D, R = []
2: repeatlet $i = 0$, $E_0$ = D* let $E_1$ Exceptional (T*, $E_0$)
3:     while $E_{i+1} \neq E_1$ do let $i = i+1$, $E_{i+1}$ = Exceptional (T*, $E_i$)
       let $D*_\infty = E_i$, T* = T* $\cup$ [$A \longrightarrow B \mid A \rightsquigarrow \mathbf{B}$], D* = D* / $D*_\infty$
4: until $D*_\infty$ = []
5: for $j = 1$ to $i$ do $D_{j-1} = E_{j-1}/E_j$, R = R $\cup$ [$D_j - 1$]
   return [T*, D*, R]
```

Finally, the third function, *querying*, will be discussed here. For this step the desired knowledge base, K*, must have already been built. There are only two scenarios that can occur when a query is made: either the query contains an element of defeasibility or it does not. If the query does not contain a defeasibile element, it can be resolved with classical datalog mannerisms, i.e. a check whether it is stored in or inferred by only the rules in T* (that is, the irrefutable facts only). More interestingly, if the query contains an element of defeasibility, the function *querying* would have to compute at what level of R (if any) would the antecedent of the defeasibile query not be considered as an exceptionality - starting from the lowest levels (i.e. most concrete rules) and moving up to increasing exceptionalities. For the purposes of this paper only the defeasibile scenario will be explained in further detail. Thus, the function *querying* takes in the elements of K* and the query, $A \rightsquigarrow B$, and returns whether T* $\models A \longrightarrow B$. The pseudo code of this function is as follows:

Algorithm 3 *Querying* (T*, $A \rightsquigarrow B$)

```
1: let $i = 0$
2: while T* $\models \bigsqcap \overline{E_i} \sqcap A \longrightarrow \bot$ and $i \leqslant n$ do let $i = i+1$
3: if $i \leqslant n$ then return T* $\models \bigsqcap \overline{E_i} \sqcap A \longrightarrow B$
4: elsereturn T* $\models A \longrightarrow B$
```

Since the nature of this algorithm allows requires recursive, the optimisations noted in (Paramá et al., 2006) [16] will be further investigated and implemented if proved to introduce significant improvements with regard to the purposes of this paper's work.

A proof that this algorithm does indeed work for all cases it is intended to is given in (Casini and Stracia, 2010) [22]. Furthermore, the computational complexity of the algorithm, as explained in (Casini et al., 2015) [?], can be proven to be EXPTIME-complete.

4 SIMILAR WORK DONE ON DEFEASIBLE DESCRIPTIVE LOGICS

In this section, work that has previously been done will be discussed - more specifically, those that are related to the work done and proposed in this paper. Furthermore, it will be clearly stated how the work done in this paper is unique and has purpose.

Many have already endeavoured to implement defeasibile reasoning into descriptive logics and ontologies [17] [18] [19] [21]. Some have even done this in datalog specifically [20] - let it be noted, however, that this has only been done for a subset of datalog and not datalog in it's entirety. The most notable contributions, however, come from (Casini and Stracia, 2010) [22] and (Casini et al., 2015) [23]. In these last two works, not only is defeasibility implemented on a theoretical level, but on a practical level as well, in the form of a concrete algorithm that is proven to work for descriptive logics.

The work in this paper differs by, not only incorporating the before mentioned algorithm into datalog in it's entirety, but also into RDFox. This has never been done before and is expected to be a great benefit to the default reasoning in artificial intelligence community.

5 CONCLUSION

A number of things have been done in this short literary review. A high level explanation and discussion was given for the concepts of classical reasoning, propositional logic and defeasibile reasoning. The same was also done for the declarative logic programming language, datalog, and the RDF store, RDFox. The need for defeasible reasoning was also made clear by means of external examples, as well as the internal Tweety-bird running example. An algorithm for the implementation of defeasibility in datalog and RDFox was also presented, explained and discussed. Finally, related work was discussed, with respect to the work done in this paper. It was also made clear that, although efforts to implement defeasibility in description logics have already been made, this has never been done for the datalog reasoner, RDFox. This will be beneficial since incorporating defeasibility with the high scalability and fast retrieval rates of RDFox is expected to significantly assist the research in exception handling within the domain and related domains of artificial intelligence. The next step of this project is to actually implement defeasibility in RDFox.

REFERENCES

[1] Pownall, T., Meyer, T. (2018). *Extending Propositional Logic with Defeasible Reasoning in Datalog - A Theoretical Approach.*

[2] Arregui, A. (2017). *Chisholm's Paradox in Should-Conditionals. SALT 27*: the University of Maryland, College Park. doi: http://dx.doi.org/10.3765/salt.v18i0.2477

[3] Prakken, H. and Sergot, M. (1995) *Contrary-to-duty obligations.*

[4] Prakken, H. and Sartor, G. (1996). A dialectical model of assessing conflicting arguments in legal reasoning. *Artificial Intelligence and Law* 4 (3-4), pp. 331-368. doi: 10.1007/BF00118496

[5] Klement, C. K.*Internet Encyclopedia of Philosophy: Propositional Logic.* Retrieved May 2018 from https://www.iep.utm.edu/prop-log/

[6] Moura, L. (2000). *Discrete Structures: Inference Rules and Proof Methods.* Retrieved May 2018 from http://www.site.uottawa.ca/~lucia/courses/2101-10/lecturenotes/04InferenceRulesProofMethods.pdf

[7] Davis, M. and Putnam, H. (July 1960). A computing procedure for quantification theory. *Journal of the ACM* 7(3) , 201–215.

[8] Zhang, H. and Stickel, M. (Feb. 2000). Implementing the Davis–Putnam Method. *Journal of Automated Reasoning* 24: 277. https://doi.org/10.1023/A:1006351428454

[9] Anderson, A. and N. Belnap [1975], *Entailment: The logic of relevance and necessity I*, Princeton: Princeton University Press.

[10] Anderson, A. and N. Belnap, and M. Dunn [1992], *Entailment: The logic of relevance and necessity II*, Princeton: Princeton University Press.

[11] Antoniou, G., Maher, M.J. and Billington, D. (June 1999). Defeasible logic versus Logic Programming without Negation as Failure. *The Journal of Logic Programming* 42, 47-57

[12] RDF Working Group. (2014). *RDF.* Retrieved May 2018 from https://www.w3.org/RDF/

[13] Nenov Y., Piro R., Motik B., Horrocks I., Wu Z., Banerjee J. (2015) RDFox: A Highly-Scalable RDF Store. In: Arenas M. et al. (eds) The Semantic Web - ISWC 2015. Lecture Notes in Computer Science, vol 9367. Springer, Cham.

[14] (March 2013) *W3C: SPARQL Query Language for RDF.* Retrieved May 2018 from ://www.w3.org/TR/rdf-sparql-query/#acknowledgements

[15] Miller, E. (2005). An Introduction to the Resource Description Framework. *Bulletin of the Association for Information Science and Technology* 25(1) pp. 15-19

[16] Paramá, J.R., Brisaboa, N.R., Penabad, M.R. et al. Acta Informatica (2006) 43: 341. https://doi.org/10.1007/s00236-006-0025-9

[17] Moodley, K., Meyer, T. and Varzinczak, I. J. (2012). A defeasible reasoning approach for description logic ontologies. In Proceedings of the South African Institute for Computer Scientists and Information Technologists Conference (SAICSIT '12). ACM, New York, NY, USA, 69-78. DOI=http://dx.doi.org/10.1145/2389836.2389845

[18] Bryant, D. and Krause, P. (2004). A review of current defeasible reasoning implementations. The Knowledge Engineering Review, Vol. 00:0,pp. 1–24, Cambridge University Press. doi: 10.1017/S000000000000000. Printed in the United Kingdom.

[19] Niemela, I. (1995). Towards efficient default reasoning. In Proceedings of the 14th international joint conference on Artificial intelligence - Volume 1 (IJCAI'95), Chris S. Mellish (Ed.), Vol. 1. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 312-318.

[20] Martinez, M. V., Deagustini, C. A. D., Falappa, M. A. and Simari, G. R. (2014). Inconsistency-Tolerant Reasoning in Datalog. In Advances in Artificial Intelligence – IBERAMIA 2014, Ana L.C. Bazzan and Karim Pichara (Eds.). Springer International Publishing, Cham, 15–27.

[21] García, A.J., and Simari, G.R. (2014). Defeasible logic programming: DeLP-servers, contextual queries, and explanations for answers. Argument & Computation, 5, 63-88.

[22] Casini, G. and Stracia, U. (2010). Rational Closure for Defeasible Description Logics. In: Janhunen T., Niemelä I. (eds) Logics in Artificial Intelligence. JELIA 2010. Lecture Notes in Computer Science, vol 6341. Springer, Berlin, Heidelberg

[23] Casini G., Meyer T., Moodley K., Sattler U., Varzinczak I. (2015). Introducing Defeasibility into OWL Ontologies. In: Arenas M. et al. (eds) The Semantic Web - ISWC 2015. Lecture Notes in Computer Science, vol 9367. Springer, Cham