Extending Propositional Logic with Defeasible Reasoning in Datalog - A Theoretical Approach

THOMAS MEYER*, University of Cape Town, South Africa
THOMAS POWNALL†, University of Cape Town, South Africa

Exception handling in reasoning logic for Artificial Intelligence is a well-known problem. In this literature review we first examine propositional logic as a basis for logic programming. We then extend propositional logic to handle exceptions to well known information, formally called defeasible reasoning. We make use of Datalog as our logic programming language because of its close fit with propositional logic in terms of declaring facts, providing rules similar to logical consequence and then querying new facts. We choose RDFox as our semantic query database as it can handle large data stores and maintain fast retrieval rates for queries. This research is divided into the implementation of *Defeasible Datalog* and the theoretical approach of maintaining propositional logic properties in *Defeasible Datalog*.

Additional Key Words and Phrases: Propositional Logic, Defeasible Reasoning, Datalog, RDFox

ACM Reference Format:
Thomas Meyer and Thomas Pownall. 2018. Extending Propositional Logic with Defeasible Reasoning in Datalog - A Theoretical Approach. 1, 1 (May 2018), 5 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Decision making is a daily occurrence in everyone's life. People must decide what to eat, what clothes to wear, what route to work they should take etc. In order for people to make these decisions it is necessary to consider certain facts. Consider the decision of what route to work someone should take. The facts that would help a person make this decision may be what the weather forecast is or news of accidents on the road. The person would consider all of these facts and decide on a particular route.

Interestingly, two people provided with the same facts may make different decisions. This shows that despite the same knowledge people may possess different reasoners within themselves to make different decisions.

The formalisation of reason has led to classical logic. This formalisation tells us of a procedure of evaluating knowledge to make decisions such that there is a single interpretation and consequently a single decision. This formalisation however useful for providing structure to decision making has its limitations. One such limitation is that of handling exceptions to our knowledge. We provide an example to highlight the research and reuse the example throughout the review.

Example x Our knowledge base will be as follows:

- All birds can fly
- A penguin is a bird
- *Follows:* A penguin can fly.

We can reason with the above knowledge base using classical logic to learn that a penguin is a bird and all birds can fly so it follows that a penguin can fly. This is reasonable given our knowledge base, if we introduce another fact that we know to be true in the real world - that penguins do not fly then we have come to an exception to our knowledge. The importance of exception handling has led to another formalised approach - that of defeasible reasoning. [10, 12]

Our project is focused on taking Datalog, a model for propositional logic, and extending it with defeasible reasoning to allow for the application of defeasible reasoners in real world models as opposed to purely theoretical formalisations. Furthermore, this project ensures that properties of propositional logic and defeasible reasoning still hold once *Defeasible Datalog* is implemented.

The motivation behind our project has been mentioned briefly. Is it the nature of decision making in our everyday life and by extension the application of decision making. We believe that our project will impact Artificial Intelligence by allowing large data stores to reason with exceptions and make logical decisions [1, 2]. Further motivation is provided by what we believe to be a gap in the literature as we have found direct evidence such a project has been completed before.

In this review section 2 provides the background for understanding the work to be done in this project. More specifically it deals with the syntax and semantics of understanding propositional logic - the chosen formal logic for this project. The notation behind defeasible reasoning and Datalog - the chosen programming language for our logic. Section 3 explains the algorithm used to extend our reasoning from propositional logic to include exception handling through defeasible reasoning. It also describes RDFox for *Defeasible Datalog*. Section 4 discusses realted work in the areas of propositional logic, defeasible reasoning and the implementation in logic programming. Lastly, section 5 concludes by emphasising the importance of the work in the context of decision making in artificial intelligence.

*Thomas Meyer is the research supervisor and student mentor.
†Thomas Pownall is the student doing this particular research as his year project for Computer Science Honours.

Authors' addresses: Thomas Meyer, University of Cape Town, South Africa, Computer Science, tmeyer@cs.uct.ac.za; Thomas Pownall, University of Cape Town, South Africa, Computer Science, pwntho001@myuct.ac.za.

XXXX-XXXX/2018/5-ART $15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

2 BACKGROUND

This section provides a brief introduction into propositional logic, defeasible logic and datalog. These topics are vastly important in understanding the research reviewed for this literature review. Without such knowledge the reader would undervalue the importance of this project.

2.1 Propositional Logic

In logic a proposition is a statement that expresses a concept that is either true or false. It follows that propositional logic focuses on applying logic to reason with propositions.

Propositions in their simplest form will be called *atomic propositions* otherwise called *atoms*. By simplest form we refer to propositions that cannot be broken down. In propositional logic we create compound propositions called *formulas* by connecting atoms by logical connectives.

The following are common logical connectives:

- *negation* $\neg$
- *disjunction* $\vee$
- *conjunction* $\wedge$
- *implication* $\implies$
- *equivalence* $\iff$

The above list of logical connectives is not complete as it does not include *exclusive or, nor* and *nand*. The missed logical connectives are not used as frequently as the above mentioned logical connectives and if required it is possible to use the above mentioned logical connectives to form the missing ones.

Example 1 Let F = {*p,q,r*} be a set of atoms. An example of a formula generated by F would be $\neg p \vee q \wedge r$.

The above example demonstrates how to create formulas from atoms, however it does not address the ambiguity of using logical connectives. The above example could be interpreted as $(\neg p \vee q) \wedge r$ or $\neg p \vee (q \wedge r)$. This ambiguity can be address by introducing precedence.

The order of precedence from high to low is:

- *negation* $\neg$
- *conjunction* $\wedge$
- *disjunction* $\vee$
- *implication* $\implies$
- *equivalence* $\iff$

We have described the syntax of propositional logic and how to create formulas from atoms. We want to define the semantics or meaning of our formulas. To do this we assign each atom in our formula with a truth value and apply the precedence and meaning of our logical connectives to determine the truth value of the formula.

Definition Let A be a formula and let $\mathcal{P}_A$ be the set of atoms appearing in A. An *interpretation* of A is a total function $\mathcal{I}_A : \mathcal{P}_A \mapsto$ {T,F} that assigns one of the truth values T or F to every atom in $\mathcal{P}_A$.

Definition Let $\mathcal{I}_A$ be an interpretation for A. $\mathcal{V}\ \mathcal{I}_A$(A), the truth value of A under $\mathcal{I}_A$ is defined recursively on the structure of A.

A formula may have a large number of interpretations and it is useful to display the interpretations. A truth table is a convenient way of doing this.

Example 2 Consider the formula $\neg p \vee q \wedge r$ from earlier. We can display this formula and all of its interpretations in a truth table.

p	q	r	$\neg$p	q $\wedge$ r	$\neg$p $\vee$ q $\wedge$ r
True	True	True	False	True	True
True	True	False	False	False	False
True	False	True	False	False	False
True	False	False	False	False	False
False	True	True	True	True	True
False	True	False	True	False	True
False	False	True	True	False	True
False	False	False	True	False	True

The last concept needed in propositional logic for our purposes is that of logical consequence also known as entailment.

Definition Let U be a set of formulas and A a formula. We write U $\vdash$ A $\iff$ every implementation that makes every formula in U true also makes A true.

Logical consequence is a powerful tool in data gathering as it allows us to take explicit facts and reason other implicit facts. We quickly mention two properties of logical consequence:

Let T be a set of formulas, let A ϵ T and B be a formula.

- *inclusion:* T $\vdash$ A $\forall$ A ϵ T
- *monotonicity:* if T' $\vdash$ B then T $\vdash$ B $\forall$ T' $\subset$ T.

Lastly, we mention the limitation of propositional logic that we wish to address. That is, propositional logic cannot handle exceptions to the atoms and formulas used in reasoning. This is best described using an example.

Example 3 Let F = {b, p, f} be a set of atoms where b,p,f represent birds, penguins and the ability to fly respectively. Let our formulas be that birds can fly b $\implies$ f and that penguins are birds p $\implies$ b. Using logical consequence it follows that penguins can fly p $\implies$ f but this is contradictory to what we know about penguins.

Ideally we would like to handle such exceptions in propositional logic without it contradicted our formulas.

2.2 Defeasible Reasoning

Defeasible reasoning is a branch of reasoning that allows for contingent statements. That is it allows statements that are subject to change. This is best described using an example, recall *Example 3*. Propositional logic is unable to handle such an exception, but defeasible reasoning can handle this. The way defeasible reasoning can

handle this exception is by introducing new notation. We introduce $\vdash$ and write A $\vdash$ B to mean that *if A then typically B*. Using this notation we can change our example to handle the exception. To do this we must find where the exception occurs, in our example this is in the statement b $\implies$ f and we change it to b $\vdash$ f.

A further classification of defeasible reasoning consequence relations is that of *rational consequence relations*. We say a consequence relation is rational $\iff$ it holds for these properties:

- *Reflexivity* C $\vdash$ C
- *Cumulative Transitivity* C $\vdash$ D and C $\wedge$ D $\vdash$ F $\implies$ C $\vdash$ F
- *Cautious Monotony* C $\vdash$ D and C $\vdash$ F $\implies$ C $\wedge$ D $\vdash$ F
- *Left Logical Equivalence* C $\vdash$ F $\vDash$ C $\iff$ D $\implies$ D $\vdash$ F
- *Right weakening* C $\vdash$ D and D $\vDash$ F $\implies$ C $\vdash$ F
- *Left Disjunction* C $\vdash$ F and D $\vdash$ F $\implies$ C $\vee$ D $\vdash$ F
- *Rational Monotony* C $\vdash$ F and C $\nvdash \neg$ D $\implies$ C $\wedge$ D $\vdash$ F

We have shown how defeasible reasoning handles exceptions and the properties of rational consequence relations, we now comment on some of the limitations and consequences of defeasible reasoning. Firstly, defeasible statements are weaker versions of their propositional counterparts and there is added complexity with ensuring defeasible statements hold the properties of the logic. Secondly, in light of the previous limitation a consequence is that we cannot nest defeasible statements. p $\vdash$ q $\vdash$ r is an example of nesting with defeasible statements and it should be clear that the statement is too weak to have meaning.

2.3 Datalog

Datalog is a declarative logic programming language that uses classical logical reasoning as its basic form of reasoning. It is often used as a query language for databases. Recently it has found new applications in a number of domains, including data integration and information extraction.

Datalog is similar to another logic programming language - Prolog. In fact, syntactically Datalog is a subset of Prolog. However, unlike Prolog, statements in a Datalog program can be stated in any order and queries on finite sets are guaranteed to terminate.

In Datalog a variable starts with an upper-case letter. A constant starts with a lower-case letter or a digit. A predicate symbol starts with a lower-case letter. A Datalog program is made up of facts, rules and queries. A fact is of the form predicateSymbol(constant,...).
Example 4

- parent(bill,tom)
- parent(tom, harry)

This tells us that bill is the parent of tom and tom is the parent of harry. A rule is of the form predicateSymbol(variable,...) :- predicateSymbol(constant,...),...
Example 5

- ancestor(X,Y) :- parent(X,Y)
- ancestor(X,Y) :- parent(X,Z),ancestor(Z,Y)

This tells us that if we know X is a parent of Y then we know that X is an ancestor of Y, it also tells us that if X is a parent of Z and Z is an ancestor of Y then X is an ancestor of Y. A query is of the form ?- predicateSymbol(constant/variable,...)
Example 6

- ?- ancestor(bill,X)

This asks who are the X that bill is an ancestor of are given our facts and rules - the answer is tom and harry.

The rules allow Datalog to infer implicit facts from explicit facts i.e. explicitly bill is a parent of tom, using our rule ancestor(X,Y) :- parent(X,Y) we can infer the implicit fact that bill is an ancestor of tom. This is very powerful since not all facts are explicitly stated or known and can allow us to reason more facts.

The format of Datalog with its facts and rules is useful for modelling propositional logic in a programming language format. Propositional statements can be written as Datalog facts and logical consequence can be written as Datalog rules. Furthermore, if we require truth values of formulas in propositional logic we can write Datalog queries to extract this information. We believe that Datalog and propositional logic are a great fit and will be able to handle the models required in our research.

3 DEFEASIBLE DATALOG

We have seen that propositional logic is useful for reasoning with formulas and deriving logical consequences. Furthermore, Datalog is a good fit for modelling propositional logic in a programming language. However, propositional logic cannot handle exceptions to formulas and by extension neither can Datalog. We have learnt that defeasible reasoning is one way in which to handle such exceptions. What we aim to do is to take propositional logic modelled for Datalog and extend it to use defeasible reasoning in order to handle the exceptions with formulas. We call this extension *Defeasible Datalog*. As previously mentioned this is a joint collaboration and my partner is handling the implementation of Defeasible Datalog. My position on this research is described below.

3.1 Theoretical Investigation

It is important that the reasoning in Defeasible Datalog must not create inconsistencies in the propositional logic at the foundation of Datalog. That is, the properties that hold in Datalog must further hold in Defeasible Datalog. My position in this research is to investigate the properties of propositional logic in Datalog and to ensure through mathematical rigour that the same properties hold in Defeasible Datalog.

The algorithm that is essential to the work done in our research is adapted from [5] low-level explanation and example to aid the reader's understanding.

3.1.1 Algorithm for extending propositional logic with defeasible reasoning. We need a way of formalising defeasible information in order to reason with new formulas without breaking propositional logic properties. We outline this formalisation in the following way. Firstly, we start with a conditional knowledge base [7, 11] K = <$\mathcal{T}$, $\mathcal{B}$> where $\mathcal{T}$ is a set of formulae representing certain knowledge and $\mathcal{B}$ is a set of sequents C $\vdash$ D representing default information.

Example 7.1 $\mathcal{T}$ = {P $\implies$ B} and $\mathcal{B}$ = {P$\vdash \neg$F, B $\vdash$F}.

We change $\mathcal{B}$ into a set of formulae to remove the defeasible information and make it formal implication. Using our *Example 7.1* we get.

Example 7.2 $\mathcal{T}$ = {P $\implies$ B} and $\mathcal{B}'$ = {(B $\implies$ F)$\wedge$(P $\implies \neg$F), P $\implies \neg$F}

Once our knowledge base is in this *default-assumption approach* it is a simple procedure to reason by only relying on logical consequence with the new formulas. Below explains the procedure of converting a conditional knowledge base into a default knowledge base. Let K = <$\mathcal{T}$,$\mathcal{B}$>.

- *Step 1:* We translate $\mathcal{T}$ into sequential form and add it to $\mathcal{B}$ to create $\mathcal{B}'$. That is we take our certain knowledge and translate it into sequents.
- *Step 2:* We define $\mathcal{T}_{B'}$ as the set of materializations of the sequents in $\mathcal{B}'$. That is we take the sequents in $\mathcal{B}'$ and make them stronger statements than they are.
- *Step 3:* We now rank the statements in terms of exceptionality. That is we perform exceptionality testing on our initial set $\mathcal{B}'$, those that are not considered exceptional have the lowest rank and those that are considered exceptional move up a ranking and the process is repeated until we get an empty set or until the same statements are repeating - if this is the case then these statements are ranked ∞.
- *Step 4:* We then determine if $\mathcal{B}'$ is inconsistent. This is determined if in its preferential closure we obtain the sequent $\top \vdash \bot$. If $\mathcal{B}'$ is consistent then we can define our background theory $\mathcal{B}'$ using those statements with rank = ∞, and we can define our set of sequents $\mathcal{B}''$.
- *Step 5:* Lastly, we translate $\mathcal{B}''$ into a set of default-assumptions. That is we have a set Δ where each element δ_i contains the sequents from rank $\rangle$ in $\mathcal{B}''$.

The above translation may be clearer through an example. Consider K = <$\mathcal{T}$,$\mathcal{B}$> from *Example 7.1.*

- Using our algorithm we translate $\mathcal{T}$ into sequential form by *Step 1* adding it to $\mathcal{B}$ to form $\mathcal{B}'$ = {P $\wedge \neg$B $\vdash \bot$, P $\vdash \neg$F, B $\vdash$ F}.
- Then we create our set of materialisations by *Step 2* $\mathcal{T}_{B'}$ = {P $\wedge \neg$B $\implies \bot$, P $\implies \neg$F, B $\implies$ F}.
- Then by *Step 3* determine our rankings: ϵ_1 = {P $\wedge \neg$B $\vdash \bot$, P $\vdash \neg$F, B $\vdash$ F}, ϵ_2 = {P $\wedge \neg$B $\vdash \bot$, P $\vdash \neg$F}, ϵ_3 = {P $\wedge \neg$B $\vdash \bot$}. Which gives the rankings r(B$\vdash$F) = 0, r(P$\vdash\neg$F) = 1 and r(P$\wedge\neg$B $\vdash \bot$) = ∞.
- *Step 4* gives our background theory $\mathcal{T}'$ = {$\neg$(P$\wedge\neg$B)}.
- By *Step 5* we get Δ = {δ_0, δ_1} with δ_0 = (B $\implies$ F)$\wedge$(P $\implies \neg$F) and δ_1 = P $\implies \neg$F.

Once we have translated our conditional knowledge base into a default knowledge base we are ready to reason with new facts and formulas. Given our default conditional knowledge base K = <$\mathcal{T}$,$\mathcal{B}$>, facts Γ and a formula D, if D is a classical formula we reason as normal, below is the procedure for reasoning with a defeasible statement D.

- We determine δ_i as the first consistent formula in the sequence of elements in Δ.
- Then we decide if our formula D follows from rational closure. We do this by determining where $\Gamma \cup \mathcal{T} \cup \{\delta_i\} \models$ D.
- If D follows from rational closure we can include it in our knowledge base, otherwise we cannot.

Take *Example 7.2* which is already a default knowledge base. We want to consider if a flying creature is not a penguin. The steps for reasoning this are as below.

- Formally write our query as: F $\vdash \neg$P
- We take the facts F and our background theory $\mathcal{T}$ and determine the first consistent formula δ_i. We determine it is δ_0.
- We cant to check if our query follows from rational closure. That is we must determine whether the following holds: F $\cup \mathcal{T} \cup \{\delta_0\} \models \neg$P.
- Since this holds we can say our query is true and F $\vdash \neg$ P.

3.2 Implementation of Defeasible Datalog

We briefly explore the implementation of Defeasible Datalog by fellow collaborator Joshua Abraham. In the previous section we describe the algorithm we intend to use to extend propositional logic in Datalog to handle defeasible reasoning. We intend to take RDFox [9] and implement the algorithm to create Defeasible Datalog. RDFox is vital to the success of this implementation and as such must be well understood by the reader.

3.2.1 RDFox: A Highly-Scalable RDF Store. Defeasible Datalog is only useful if it has applications. It is not difficult to find such applications and the application we focus on is that of the Semantic Web. There is a vast quantity of knowledge represented in many formats across the Web. We focus on knowledge using the Resource Description Framework (RDF) - such knowledge is typically referred to as RDF stores which is a specialised database of triples. These triples are of the form *subject-predicate-object* and our RDF store should also allow for their retrieval through semantic queries.

RDFox is the RDF store we have decided to use and the reason for this is its ability to handle large amount of data and maintain quick retrieval rates [9]. These properties are essential since the Semantic Web is so large and an RDF Store that cannot handle large amounts of data has little usability in such an environment. Another reason for our choice of RDFox is that its reasoning for query answering is Datalog, which is the language with which we are extending propositional logic.

4 RELATED WORK

However important we believe our research to be, it is important to review the literature of what has already been done to establish the importance of our research to the academic world. The ability to handle exceptions and reason with incomplete information is at the heart of Artificial Intelligence. It is first documented on how to handle such exceptions, one of the results being defeasible reasoning. With the formalisation of defeasible reasoning followed many

extensions of formal logic to incorporate it. In particular, the extension of description logic. [3, 4, 6] As well as extensions to formal logic, we have seen logic programming languages implementing propositional logic for querying and reasoning.[9]. We narrowed our research to formal logics with Datalog and found attempts to extend its reasoning with defeasible reasoning [8**?**] but we note that their research is on a subset of Datalog. We further motivate our research by stating that we are focused on extending Datalog in its entirety with defeasible reasoning. As well as demonstrating the value of an applicable implementation by using RDFox for large data storage and quick retrieval.

5 CONCLUSION

The importance of exception handling is at the heart of recent Artificial Intelligence. Defeasible reasoning is a solution that has gained a lot of attention and as such there is much research into extending formal logic using it. Furthermore, these extensions are unimportant without application. The review has found many attempts to extend logic programming languages to incorporate defeasible reasoning. However, we have found no evidence to suggest the proposed research is in the literature. That is, there is no evidence to suggest research into extending the logic programming language Datalog with defeasible reasoning in RDFox.
This is promising as implementing a defeasible logic programming language - *Defeasible Datalog* with the scalability and fast retrieval rates of RDFox will benefit research in exception handling within Artificial Intelligence greatly.

REFERENCES

[1] 1993. *Handbook of logic in artificial intelligence and logic programming*. Clarendon Press, Oxford.
[2] P.A. Bonatti, M. Faella, and L. Sauro. [n. d.]. ([n. d.]).
[3] Giovanni Casini, Thomas Meyer, Kody Moodley, Uli Sattler, and Ivan Varzinczak. 2015. Introducing Defeasibility into OWL Ontologies. In *The Semantic Web - ISWC 2015*, Marcelo Arenas, Oscar Corcho, Elena Simperl, Markus Strohmaier, Mathieu d'Aquin, Kavitha Srinivas, Paul Groth, Michel Dumontier, Jeff Heflin, Krishnaprasad Thirunarayan, and Steffen Staab (Eds.). Springer International Publishing, Cham, 409–426.
[4] G. Casini, T. Meyer, K. Moodley, and I. Varzinczak. [n. d.].
[5] Giovanni Casini and Umberto Straccia. 2010. Rational Closure for Defeasible Description Logics. In *Logics in Artificial Intelligence*, Tomi Janhunen and Ilkka Niemelä (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 77–90.
[6] L. Giordano, V. Gliozzi, N. Olivetti, and G.L. Pozzato. 2013. A non-monotonic Description Logic for reasoning about typicality. *Artificial Intelligence* 195 (2013), 165 – 202. https://doi.org/10.1016/j.artint.2012.10.004
[7] Daniel Lehmann and Menachem Magidor. [n. d.]. ([n. d.]).
[8] Maria Vanina Martinez, Cristhian Ariel David Deagustini, Marcelo A. Falappa, and Guillermo Ricardo Simari. 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.
[9] Yavor Nenov, Robert Piro, Boris Motik, Ian Horrocks, Zhe Wu, and Jay Banerjee. 2015. RDFox: A Highly-Scalable RDF Store. In *The Semantic Web - ISWC 2015*, Marcelo Arenas, Oscar Corcho, Elena Simperl, Markus Strohmaier, Mathieu d'Aquin, Kavitha Srinivas, Paul Groth, Michel Dumontier, Jeff Heflin, Krishnaprasad Thirunarayan, and Steffen Staab (Eds.). Springer International Publishing, Cham, 3–20.
[10] John L. Pollock. 1987. Defeasible reasoning. *Cognitive Science* 11, 4 (1987), 481 – 518. https://doi.org/10.1016/S0364-0213(87)80017-4
[11] R. Reiter. 1980. A logic for default reasoning. *Artificial Intelligence* 13, 1 (1980), 81 – 132. https://doi.org/10.1016/0004-3702(80)90014-4 Special Issue on Non-Monotonic Logic.
[12] P. Simons and Nuel D. Belnap. 1995. *Studia Logica: An International Journal for Symbolic Logic* 54, 2 (1995), 261–266. http://www.jstor.org/stable/20015782