Literature Review on Justifications for Classical Entailment

Orefile Morule
MRLORE001@uct.ac.za
University of Cape Town
Cape Town, South Africa

ABSTRACT

Justifications help knowledge base users and engineers to understand why a given given entailment holds. Having this information is beneficial in understanding the knowledge represented in the system and how this was used to infer new knowledge. There are several algorithms used get find justifications and they each have their advantages and disadvantages. Justifications can be fine-grained in order to remove superfluity and make more sense of why an entailment holds.

CCS CONCEPTS

• **Theory of computation → Automated reasoning**; • **Computing methodologies → Nonmonotonic, default reasoning and belief revision**; *Description logics.*

KEYWORDS

artificial intelligence, knowledge representation and reasoning, classical reasoning

1 INTRODUCTION

Formal logic in the field of *Artificial Intelligence* allows us to represent the state of the world in such a way that it is easy for automated systems to understand it. Once this is achieved we would like the systems to be able to infer new knowledge by reasoning with that it already knows. This is the concept of *Knowledge Representation and Reasoning* in Artificial Intelligence. We would like to know how and why Artificial Intelligence systems come to conclusions when asked if something holds in the world as it knows it. This is known as *Justifications for entailment.*

We will start this paper by giving background on *Propositional Logic* which is the formal logic this paper will be restricted to. We will then show how logic ties in with Knowledge Representation and Reasoning. Then we will explain Justifications for Classical Entailment including the different algorithms for computing justifications and the benefits and disadvantages associated with them. We will show how the *black-box* algorithm can be optimised to reduce the number of entailment tests that need to be run.

Finally, in the discussion we will explain how *Fine-grained justifications* can be used to solve issues of *superfluity* and *masking*. Furthermore we will discuss how different justification computing formulas can be used in combination.

2 BACKGROUND

2.1 Propositional Logic

Propositional Logic is the branch of logic concerned with joining and/or modifying statements to create more complex statements [7]. Propositional logic has indivisible statements referred to as *atoms*. Atoms have a truth value - either True or False, but not both, and represent what we know about the world. An example would be, "the sky is blue," or "it is raining."

2.1.1 ***Syntax***. The set of statements or atoms is denoted as $\mathcal{P}$. In the language of Propositional Logic, atoms are represented as lowercase letters. A statement such as, "the sky is blue," could then be represented as s. Atoms are joined or modified using logical signs or *connectives* to create more complex statements called *formulas*. The main connectives are *conjunction*, *disjunction*, *negation*, *implication*, and *equivalence*. They are used in place of the truth-functional operators, *"and"*, *"or"*, *"not"*, *"if… then"*, and *"if and only if"*, respectively. The negation connective is unary, meaning it takes only one operand, while the rest are binary, meaning that they take on two operands.

See the table below for connectives and their symbols.

Name	Meaning	Symbol
disjunction	or	$\vee$
conjunction	and	$\wedge$
negation	not	$\neg$
implication	if then	$\rightarrow$
equivalence	if and only if	$\leftrightarrow$

Figure 1: The Boolean operators used in propositional logic

The formula $p \rightarrow q$ is read "p implies q" and means if p is True then q is True. The set of all formulas can be denoted as $\mathcal{L}$ such that for every $p \in \mathcal{P}, p \in \mathcal{L}$, and if $\alpha, \beta \in \mathcal{L}$, then $\neg\alpha, \alpha \wedge \beta, \alpha \vee \beta, \alpha \rightarrow \beta, \alpha \leftrightarrow \beta \in \mathcal{L}$ [18].

2.1.2 ***Semantics***. A *valuation* V is a function used to assign a Truth value to an *atom*. A *truth table* is a finite table used to show each possible combination of Truth values of atoms. Each line in a truth table corresponds to a different valuation [7]. A proposition having a Truth value True, is called *Satisfaction*. Formally, A *Valuation* V satisfies some $p \in \mathcal{P}$ if $V(p) = T$ and it is denoted by $V \Vdash p$. A *valuation* does not satisfy some $p \in \mathcal{P}$ if $V(p) = F$ and it is denoted by $V \nVdash p$. In order to derive the Truth value of a propositional formula under any valuation V, we look at the Truth vales of the individual atoms making up the formula and use the semantics and precedence order of the connectives used to evaluate whether the formula is True or False.

V is a *model* of α if for any $\alpha \in \mathcal{L}$ and a *valuation* V, $V \Vdash \alpha$.

2.2 Knowledge Representation and Reasoning

In Artificial Intelligence, we want to represent knowledge about the world in such a way that a machine can understand reason about the representation [1]. Formal logic is used to represent knowledge by the use of symbols to represent propositions and propositional

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	F	F	F	T	T

Figure 2: A truth table for the Boolean operators $\neg$, $\wedge$, $\vee$, $\rightarrow$ and $\leftrightarrow$

formulas. A key component in *Knowledge Representation* is a *Knowledge Base*. A *Knowledge Base* is a finite set of propositional formulas [18].

Satisfiability for a *Knowledge Base* $\mathcal{K}$ means that for every $\alpha \in \mathcal{K}$ there exists a *valuation* V that satisfies each of them.

Reasoning is the process of using the knowledge represented in a knowledge base to make inferences and make decisions. The reasoning technique this paper focuses on is *deduction* where a conclusion logically follows from the premises given.

3 JUSTIFICATIONS FOR ENTAILMENT

Inferring new information using knowledge from a knowledge base is *entailment*. Formally, if we let $U \subseteq \mathcal{L}$ be a set of formulas and α be a formula, U *entails* α ($U \models \alpha$), if and only if every model of U is a model of α.

Satisfaction provides a way of checking entailment. $\mathcal{K} \models \alpha$ if $\mathcal{K}$ with $\neg\alpha$ added to it is not satisfiable.

It is beneficial to know how an automated system comes to a conclusion. AI systems are being used in essential parts of our lives including healthcare, transport, and security [17]. *Explanations* provide reasons as to why an inference was made. This review focuses on a specific type of explanation called *Justification*, which is the minimal subset of formulas needed for the entailment to hold [2].

Let us demonstrate this concept with an example:

(1) Birds fly
(2) Birds have wings
(3) Tweety is a bird

The statements above are our propositions. If someone were to ask, 'does Tweety fly?' all we would need to answer that are the statements, 'Birds fly' and 'Tweety is a bird'. The statement 'Birds have wings' is irrelevant in our justification because even without the statement, our entailment holds. Conversely, the entailment would not hold if we did not have the statements 'Birds fly' and 'Tweety is a bird'.

We can define justification in relation to Knowledge Bases. Formally, if we Let $\mathcal{K}$ be a knowledge base and have a query α such that $\mathcal{K} \models \alpha$; the set of formulas J is a justification for $\mathcal{K} \models \alpha$ if $J \subseteq \mathcal{K}$, $J \models \alpha$ and for all $J' \subset J$, it holds that $J' \not\models \alpha$ [18].

3.1 Justification algorithms

The examples we have looked show only a handful of propositional formulas, but in reality knowledge bases can contain thousands of formulas. It is thus necessary to have algorithms that can compute justifications for entailments in the most efficient way. In the literature, algorithms for computing justifications are broadly classified into two categories, *black-box algorithms* and *glass-box algorithms* [8]. Horridge [2] goes further by stating that there are two axes of classification for algorithms for computing justifications. The first axis deals with whether an algorithm computes a single or all justifications for an entailment and this is called the *single-all-axis*. The second axis deals with whether the algorithm is a black-box or glass-box algorithm and this is called the *reasoner-coupling-axis*.

3.1.1 ***Single-All-Axis.*** Algorithms for computing all justifications tend to depend on algorithms for computing a single justification as a sub-routine [2]. Computing a single justification can be beneficial for debugging a knowledge base. If an engineer for a knowledge base cannot make sense of an entailment, it would be easier to find the error in the knowledge base if they get a single justification at a time.

3.1.2 ***Reasoner-Coupling-Axis.*** Penaloza et al [11], explain that the glass-box algorithm aims to keep track of the propositional formulas being used in the reasoning algorithm. This requires modifying the reasoner to ensure it does not only state whether or not an entailment holds but which propositional formulas or justification make the entailment hold [8].

Contrastingly, the black-box algorithm is independent of the reasoning algorithm and does not need to modify its internal workings. We can think of the reasoner as a black-box that accepts a knowledge base $\mathcal{K}$ and an entailment $\mathcal{K} \models \beta$ as input and returns a yes or no depending on whether or not the entailment holds [6]. The black-box algorithm consists of two phases, *expand* and *contract*. During the expansion phase, a subset S of the knowledge base is expanded with propositional formulas until the entailment holds in the subset. At this point either the subset S or some subset of S is guaranteed to be a justification for $\mathcal{K} \models \beta$. We now enter the contraction stage where propositional formulas are removed from the subset S until it is a minimal set of propositional formulas that entails β. Both stages require satisfiability checks to the reasoner [6].

The most obvious way to go about expansion is to start with a subset S that is equal to the knowledge base $\mathcal{K}$. The Contraction stage will then remove one formula at a time until each formula has been checked. At this point S is a justification for $\mathcal{K} \models \beta$. The number of tests needed would be proportional to the size of $\mathcal{K}$. As mentioned previously, knowledge bases in reality contain thousands of formulas, therefore it would be impractical to use such a method.

3.2 Optimising Expansion phase

The aim of optimising the expansion phase is to have as small a number of entailment tests as possible. One of the techniques used for this is the *selection function*. A *selection function* γ, creates all possible subsets of $\mathcal{K}$. This reduces the number of tests done in the expansion phase. The chances of selecting S where $J \subset S$ increases with an increasing size of J. In the best case scenario, only a single test such as $S \models \beta$ has to be done [9]. Horridge [2] explains that expansion tries to find the entailment in question in a subset of the knowledge base much smaller than the knowledge base. The benefit of this is that entailment tests are faster for smaller sets

and a smaller input to the contraction phase typically means fewer entailment tests in that phase.

3.3 Optimising Contraction phase

There are two optimisations that can be implemented in the contraction phase. The first is the *sliding window* approach. Instead of choosing one formula to remove from the subset S, a set of formulas S" is removed from S. If $S \setminus S" \models \beta$, then $n = |S"|$ number of formulas can be removed from S. This will result in having n - 1 entailment tests as opposed to n entailment tests. The second approach is a *divide and conquer* strategy. Given a Knowledge Base $\mathcal{K}$, where $\mathcal{K} \models \beta$, $\mathcal{K}$ is split into two halves $\mathcal{K}1$ and $\mathcal{K}2$. If $\mathcal{K}1 \models \beta$ then $\mathcal{K}2$ is discarded and vice versa. If neither $\mathcal{K}1$ nor $\mathcal{K}2$ entails β, then $\mathcal{K}1$ is split into half and one of those halves is combined with $\mathcal{K}2$ and an entailment test is ran on that combination. The process of halving and merging is continued until a justification is obtained. Shchekotykhin et al [16] carried out a comparison between the sliding window strategy used by Kalyanpur et al [6] and the dividing and conquer strategy, and found that the divide and conquer strategy performed noticeably better [2].

3.4 Computing all justifications

A *Hitting Set Tree* [12] can be used to compute all justifications in a knowledge base. Given $\mathcal{K} \models \beta$, a hitting set tree for β in $\mathcal{K}$ is a finite tree which has nodes that are marked with justifications for $\mathcal{K} \models \beta$, and edges marked with formulas contained in $\mathcal{K}$. Each non-leaf node v is connected to a child node v' through an edge marked with a formula α such that α is in the mark of v but not in the mark of v'. v' is a leaf node if the mark of v' is an empty set.

Here are the steps for building the tree:

(1) We start by choosing a formula $\alpha \in v$ which is not in the edges of the children of node v.
(2) We then let S be the disjunction of the chosen formula α and the set of formulas found in the path from v to the root node of the tree. We then remove S from $\mathcal{K}$ to get $\mathcal{K}'$
(3) We then run an entailment test to check if the new knowledge base $\mathcal{K}' \models \beta$. If so, we compute a justification J for β with respect to $\mathcal{K}$. If instead $\mathcal{K} \not\models \beta$, we set the justification $J = \emptyset$.
(4) We then make a new node v' and set its justification to be J
(5) We add an edge e that connects v and v', and we label e with the formula α
(6) Finally we add all the formulas in S back into $\mathcal{K}$

The enumerated procedure is repeated until no new child nodes can be created. At this stage all the justifications for $\mathcal{K}' \models \beta$ can be found in the node labels.

3.5 Fine-grained Justifications

Horridge [3] states that there are cases where not all parts of a formula are needed for an entailment to hold. He then presents the idea of a *fine-grained justification*, which is a justification that contains formulas with no *superfluous* parts. For a justification J for an entailment β, an atom that has a Truth value **T** is regarded as being superfluous if it can be substituted for another positive atom without breaking β, and an atom that has a Truth value **F** is regarded as being superfluous if it can be substituted for another negative atom without breaking β.

Consider the following example:

(1) Birds fly
(2) Birds have wings
(3) Penguins are birds and are cute

If we have a query, *'do penguins fly'*, our justification would be *'Penguins are birds and are cute', 'Birds fly'*. Intuitively, one can see that the information about penguins being cute is not relevant as the statement *'Penguins are birds and are cute'* can be replaced with *'Penguins are birds and are awesome'* and the entailment would still hold.

Fine-grained justifications are further split into *laconic* and *precise* justifications. Laconic justifications are justifications whose formulas do not contain superfluous parts and precise justifications are justifications that identify the parts of formulas which could be changed for *repair* of a knowledge base. Formally, given $\mathcal{K} \models \beta$, the set of formulas R is a repair for β in $\mathcal{K}$ if $R \subseteq \mathcal{K}, \mathcal{K} \setminus R \not\models \beta$ and there is no $R' \subsetneq R$ such that $\mathcal{K} \setminus R' \not\models \beta$.

4 DISCUSSION

The idea of justifying entailments has been laid out in this review. The main reasons one would want to seek a justification for an entailment are (1) debugging an erroneous knowledge base, (2) exploring the facts about the world stored in the knowledge base, and (3) understanding the complexity of the knowledge base. A lot of the work cited has been done by Horridge. He states that explanations were initially mostly *proof-based*. That is to say an explanation showed how a reasoner **proved** that entailment held in a knowledge base. Work by Schlobach and Cornet [15] and [10] is noted as being a significant cause to shifting attention towards justification-based explanations. This means that in order to understand a justification, one does not need to know how proofs work and the rules used for deduction. Instead all one needs is to understand the semantics of propositional logic. The literature covers justification-based explanations in terms of *ontologies* and not knowledge bases. An ontology is essentially a formal representation of a domain of knowledge. It is related to knowledge bases in that an ontology can be the foundation upon which a knowledge base is built and that allows us to extend the concepts of justification as well as knowledge representation and reasoning to knowledge bases.

The different algorithms for computing were shown along the single-all axis and the reasoner-coupling axis. The algorithms can be combined [14]. One combination sometimes referred to as the *pure black-box algorithm* is where you have a black-box algorithm for computing all justifications using a black-box algorithm for computing a single justification as a sub-routine [5]. Another combination sometimes referred to as *hybrid black-box glass-box algorithm* is where you have a black-box algorithm for computing all justifications using a glass-box algorithm for computing single justifications as a subroutine. Lastly we have what is sometimes referred to as *pure glass-box algorithm* where you have a glass-box algorithm for computing all justifications using a glass-box algorithm for computing a single justification as a sub-routine.

Black-box algorithms have the advantage of easy implementation due to being independent of the reasoner. The disadvantage of a black-box algorithm is that it can be impractical for a large

knowledge base where it might need to expand over a large portion of the knowledge base. Glass-box algorithms have the advantage of being able to compute single justifications quickly and trivially as part of the reasoning process [2] [13]. A disadvantage of glass-box algorithms is that they are difficult to implement as modifications to the reasoner need to be made. Additionally they have a disadvantage as compared to black-box algorithms of more memory usage [13].

A problem that has been highlighted with justifications is superfluity. This concepts refers to formulas in a justification having unnecessary parts for the entailment to hold. A related concept that was introduced is masking, which means that the number of justifications for an entailment do not equal the number of reasons for that entailment. Horridge [4] explains that there are two main problems with masking. The first is that because there can be multiple reasons for an entailment within a single justification, this can hinder one's comprehension of why the entailment holds. Secondly if a knowledge base engineer needs to repair the knowledge base by removing parts of formulas, then the repair might not give the desired result because all the reasons for an entailment might not be clear. As a solution to superfluity and masking Horridge [2] proposes fine-grained justifications. These are categorised into laconic and precise justifications. Having fine-grained justifications allows one to know which parts of the formulas in a justification are needed for an entailment to hold. This aids in terms of comprehension of the justification of an entailment and also helps in terms of making a successful repair.

5 CONCLUSIONS

We have looked at how propositional logic allows us to represent statements about the world using symbols and connectives between them. This is useful in terms of Artificial Intelligence because it makes it easier for a computer to understand this representation and reason about it. We then explained that one would want to understand why an entailment from a knowledge base holds for reasons including that the automated decision can be in fields including healthcare and security where there is little to no margin of error.

We looked at the black-box and glass-box algorithms and how they are used to compute justifications. The main difference between them is that black-box algorithms are independent of the reasoner whereas glass-box algorithms need to be integrated with a reasoner in order to compute justifications. The black-box algorithms can be optimised to reduce the number of entailment tests needed at each step by using a selection function in the expansion phase and during the contraction phase using either a sliding window approach or a divide and conquer strategy. Shchekotykhin et al [16] showed that the divide and conquer performs better than the sliding window approach. We also noted that there can be more than one justification for an entailment. The Hitting Set Tree is an algorithm that is most common for computing all the justifications for an entailment.

Finally we have explained that justifications have a problem where they can be superfluous and that one could have justification masking. A way to circumvent this is by having fine-grained justifications. Horridge's work [2] defines that fine-grained justifications can be laconic or precise. Our discussion showed that this helps with comprehending why an entailment holds and how to go about knowledge base repairs for the desired result.

REFERENCES

[1] Crina Grosan and Ajith Abraham. 2011. *Knowledge Representation and Reasoning.* Springer Berlin Heidelberg, Berlin, Heidelberg, 131–147. https://doi.org/10.1007/978-3-642-21004-4_6

[2] Matthew Horridge. 2011. *Justification based explanation in ontologies.* Ph. D. Dissertation. University of Manchester UK.

[3] Matthew Horridge, Bijan Parsia, and Ulrike Sattler. 2009. Computing explanations for entailments in description logic based ontologies. In *16th Automated Reasoning Workshop (ARW 2009).*

[4] Matthew Horridge, Bijan Parsia, and Ulrike Sattler. 2010. Justification masking in OWL. In *23rd International Workshop on Description Logics DL2010.* 32.

[5] Qiu Ji, Peter Haase, Guilin Qi, Pascal Hitzler, and Steffen Stadtmüller. 2009. RaDON—repair and diagnosis in ontology networks. In *The Semantic Web: Research and Applications: 6th European Semantic Web Conference, ESWC 2009 Heraklion, Crete, Greece, May 31–June 4, 2009 Proceedings 6.* Springer, 863–867.

[6] Aditya Kalyanpur, Bijan Parsia, Matthew Horridge, Evren Sirin, et al. 2007. Finding all justifications of OWL DL entailments. *ISWC/ASWC* 4825 (2007), 267–280.

[7] Kevin C. Klement. 2004. Propositional Logic. In *Internet Encyclopedia of Philosophy.*

[8] Michel Ludwig. 2014. Just: a Tool for Computing Justifications wrt ELH Ontologies. *ORE* 1207 (2014), 1–7.

[9] Solomon Malesa. [n. d.]. Explanations in Logic-Based Reasoning Systems: A theoretical framework for generating explanations. ([n. d.]).

[10] Bijan Parsia, Evren Sirin, and Aditya Kalyanpur. 2005. Debugging OWL ontologies. In *Proceedings of the 14th international conference on World Wide Web.* 633–640.

[11] Rafael Penaloza and Baris Sertkaya. 2010. Complexity of Axiom Pinpointing in the DL-Lite Family of Description Logics.. In *ECAI*, Vol. 215. 29–34.

[12] Raymond Reiter. 1987. A theory of diagnosis from first principles. *Artificial intelligence* 32, 1 (1987), 57–95.

[13] Patrick Rodler. 2016. Interactive debugging of knowledge bases. *arXiv preprint arXiv:1605.05950* (2016).

[14] Patrick Rodler. 2022. One step at a time: An efficient approach to query-based ontology debugging. *Knowledge-Based Systems* 251 (2022), 108987.

[15] Stefan Schlobach, Ronald Cornet, et al. 2003. Non-standard reasoning services for the debugging of description logic terminologies. In *Ijcai*, Vol. 3. 355–362.

[16] Kostyantyn Shchekotykhin, Gerhard Friedrich, and Dietmar Jannach. 2008. On computing minimal conflicts for ontology debugging. *Model-Based Systems* 7 (2008).

[17] Warren J von Eschenbach. 2021. Transparency and the black box problem: Why we do not trust AI. *Philosophy & Technology* 34, 4 (2021), 1607–1622.

[18] S Wang. 2022. *Defeasible Justification for the KLM Framework.* Master's thesis.