Propositional Typicality Reasoning (PTR)

An Implementation of LM-Entailment as a form of reasoning for Propositional Typicality Logic (PTL)

Guy Green
University of Cape Town
Department of Computer Science
guygreen1995@gmail.com

ABSTRACT

Propositional Typicality Logic (PTL) is a recently proposed form of logic. It extends Classical Propositional Logic by the addition of Typicality with the typicality operator (•). The typicality operator is based on and extends the work done in the KLM-Approach with the defeasibility operator ($\mid\!\sim$) first suggested by Kraus et al [9]. The extension is that the typicality operator is an unary operator, while the defeasibility operator is a binary operator.
There have been two forms of entailment for PTL suggested by Booth et al [2]: they are LM and PT Entailment. Both of these forms of entailment are extensions of the Rational Closure Construction and Algorithm first shown in Lehmann and Magidor [11] for Defeasibile Reasoning. Both forms of entailment focus on satisfying a different subset of properties.
This paper is a proof of concept of an implementation of LM Entailment, reducing the entailment problem in PTL to a series of Classical Entailment checks which are effected by a SAT-Solver. The reduction is achieved by performing the algorithm put forward for LM-Entailment to make the LM Preference Relation, then converting this ranked model to be compatible with the SAT-Solver.
This paper shows a successful proof of concept of an LM-Entailment implementation, having been successfully tested with sample knowledge bases. Possible future work could include producing sample knowledge bases to test all the properties of entailment in PTL and to theoretically produce the Ranked Models that LM Entailment produces with these knowledge bases. Working on the efficiency of the implementation is also a possibility, as this was not focused on in this paper

KEYWORDS

Propositional Logic, Defeasibility, Defeasible Reasoning, Typicality, Entailment, Rational Closure

1 INTRODUCTION

Propositional Typicality Logic (PTL) is a (newly) proposed logic[4], which is an extension of Classical Propositional Logic. It extends Classical Propositional Logic by adding the Typicality Operator (•). The intended purpose of adding the notion of typicality to Classical Propositional Logic would be to expand for cases in which the sentence could be false without making the model inconsistent. The most typical cases, the sentence will be true, but otherwise it will not, allowing more expressibility and better exception handling. In the example of the *tweety* knowledge base, it allows for sentences:

- Typical birds fly: $\bullet b \rightarrow f$
- Penguins are birds: $p \rightarrow b$
- Typical penguins don't fly: $\bullet p \rightarrow \neg f$

In this knowledge base, it would concluded that Penguins are not typical birds. The equivalent knowledge base in Classical Propositional Logic would be:

- Birds fly: $b \rightarrow f$
- Penguins are birds: $p \rightarrow b$
- Penguins don't fly: $p \rightarrow \neg f$

In this knowledge base, it would be concluded that there are no penguins, which does not necessarily give an accurate reflection of what is being modeled.
PTL is built upon the work done in Defeasible Reasoning with the addition of the defeasibility operator ($\mid\!\sim$). This is called the KLM-Approach [9]. It extends this work by allowing the typicality operator anywhere in the sentence. This is not the case for the defeasibility operator because it is binary, whereas the typicality operator is unary.
Because the operator is unary, PTL adds more expressibility than the KLM-Approach. This is important for a number of reasons: it puts forward a relatively simple logic in terms of number of operators; it is very expressive; and it provides a more accurate modeling of situations.
Despite not having many operators, difficulties do arise with regards to entailment and reasoning due to the extra expressibility. Booth et al [2] proposed two different methods for entailment in PTL: LM and PT entailment. These are both extensions of the Rational Closure Construction and Algorithm initially suggested for reasoning for the KLM-Approach (Lehmann and Magidor [11]).
This paper focuses on a proof of concept of the implementation of Entailment in PTL, focusing on LM-Entailment, while my project partner, Andrew Howe-Ely, focuses on a proof of concept of the implementation of PT-Entailment.

2 BACKGROUND

2.1 Classical Propositional Logic

Classical Propositional Logic forms the basis of all other logics. It is made up of boolean operators and propositional atoms [1]. Propositional atoms are generally represented by lower-case letters and either have a value of true or false.
In the context of logics, the assigning of meaning to a symbol is called an interpretation. In propositional logic, this is just assigning a true or false value to an atom. Interpretations refer to each atom in a given knowledge base having been assigned either a true or false value. The concept of interpretations is vital in later sections, especially with regards to reasoning and entailment in the different logics.
The important boolean operators in Propositional logic are:

- Implication ($\rightarrow$)

- Disjunction/Or ($\vee$)
- Conjunction/And ($\wedge$)
- Equivalence ($\leftrightarrow$)
- Negation/Not ($\neg$)

All of these operators are binary operators, with the exception of the negation operator, which is unary. The left hand side of a binary boolean operator is generally referred to as the antecedent and the right hand side, the consequence. Combinations of these boolean operators and propositional atoms are built up to make up sentences which can be used to describe situations or rules which are to be modeled. Sentences can be valued as either true or false relative to interpretations. Groups of sentences can be made that describe a situation in as much detail as possible. A group of sentences grouped together to describe a specific situation is known as a Knowledge Base ($\mathcal{K}$).

Entailment is the crucial part of reasoning in logics. Interpretations are fundamental to the idea of reasoning. Sentences are called satisfiable if there is a minimum of one interpretation in which the sentence is true. To assess whether or not a sentence entails from a knowledge base, the first step is to produce the set of interpretations that make the conjunction of all the sentences in the knowledge base satisfiable. We call the set of interpretations that makes the conjunction of a set of sentence(s) the model of that set of sentence(s). The second step is to take the model of the sentence that is being checked (whether or not it entails from the knowledge base). It can be said that a sentence, α, entails from a knowledge base, $\mathcal{K}$, ($\mathcal{K} \models \alpha$) if and only if the model of the knowledge base is a subset of the model of the sentence ($Mod(\mathcal{K}) \subseteq Mod(\alpha)$).Entailment in Classical Propositional Logic can be reduced to a Boolean-Satisfiability problem.

The Boolean-Satisfiability Problem[8] is given a boolean formula, is there an interpretation that exists that satisfies it. It is often referred to as the SAT Problem. This is exactly the question that is being asked in entailment. This problem is a known NP-complete problem [6]. SAT-Solvers have been created to solve this problem. In this paper SAT-Solvers are used to help with the implementation of LM-Entailment.

2.2 Defeasible Reasoning

Classical Propositional Logic is not capable of dealing with exceptions as has been shown in the *tweety* example. A way to fix this problem is the addition of a binary operator called the Defeasibility Operator($\mid\!\sim$) to Classical Propositional Logic - also called the KLM-Approach - first introduced by Kraus, Lehmann and Magidor[9]. This allows for a sentence to have exceptions.

Kraus, Lehmann and Magidor [9] also discuss the properties (shown below) that defeasibility operator has that make it a preferential conditional. It also satisfies the property of Rational Monotonicity (RM; shown below) and so is said to be rational conditional.

$$\text{(Ref)} \quad \alpha \mid\!\sim \alpha \qquad\qquad \text{(LLE)} \quad \frac{\models \alpha \leftrightarrow \beta,\ \alpha \mid\!\sim \gamma}{\beta \mid\!\sim \gamma}$$

$$\text{(And)} \quad \frac{\alpha \mid\!\sim \beta,\ \alpha \mid\!\sim \gamma}{\alpha \mid\!\sim \beta \wedge \gamma} \qquad \text{(Or)} \quad \frac{\alpha \mid\!\sim \gamma,\ \beta \mid\!\sim \gamma}{\alpha \vee \beta \mid\!\sim \gamma}$$

$$\text{(RW)} \quad \frac{\alpha \mid\!\sim \beta,\ \models \beta \rightarrow \gamma}{\alpha \mid\!\sim \gamma} \qquad \text{(CM)} \quad \frac{\alpha \mid\!\sim \beta,\ \alpha \mid\!\sim \gamma}{\alpha \wedge \beta \mid\!\sim \gamma}$$

$$\text{(RM)} \quad \frac{\alpha \mid\!\sim \gamma,\ \alpha \not\mid\!\sim \neg\beta}{\alpha \wedge \beta \mid\!\sim \gamma}$$

An example of using the KLM-Approach is as follows: In the knowledge base that contains the atoms, $\{p, b, f\}$, where p, b and f stand for penguins, birds and fly respectively. The sentence $b \mid\!\sim f$ says typically if birds is true then fly (or can fly) is true but it also allows for the exception that a penguin is a bird that cannot fly

With this addition, entailment and reasoning become more complex. Lehmann and Magidor [11] suggest the concept of *Ranked Interpretations* - an ordered structure of interpretations ranked on their typicality. This allows for the concept of entailment and reasoning to be possible with regards to the defeasibility operator.

The Rational Closure Construction and Algorithm was suggested [11] as the primary method to produce the Ranked Interpretation structure and check entailment in Defeasible Reasoning. They also showed that the Rational Closure Construction and Algorithm produces a minimal ranked model that represents the most typical situation of the world. The concept of a minimal ranked model is crucial in later sections. Slight adaptions to the concept of Ranked Interpretations have been made by Booth and Paris [5] and Giordano et al. [7] Once we have this ranked interpretation structure, entailment in the KLM-Approach can be reduced to a Boolean Satisfiability Problem.

In the *tweety example* in KLM-Approach, the knowledge base is as follows:

- $b \mid\!\sim f$ (Birds typically fly)
- $p \rightarrow b$ (Penguins are birds)
- $p \mid\!\sim \neg f$ (Penguins typically do not fly)

The Rational Closure Construction and Algorithm produces the Ranked interpretation structure shown in Figure 1. The first value represents the boolean value for penguin. The second value represents the boolean value for bird and the third value represents the boolean value for flying.

L_∞ :	100		101
L_2 :		111	
L_1 :	010		110
L_0 :	000	001	011

Figure 1: The Ranked Interpretation structure for the *tweety* example

2.3 Propositional Typicality Logic (PTL)

Propositional Typicality Logic (PTL) is newly proposed extension to Classical Propositional Logic. PTL is Classical Propositional Logic with the addition of the Typicality Operator ($\bullet$). While the typicality and defeasibility operators are fairly similar, a major difference lies in the former's being unary, which allows it to be anywhere in the sentence. The defeasibility operator, being binary, only allows for typicality on the antecedent.

An example of the typicality operator being used is as follows: in the knowledge base that contains the atoms, $\{pa, p, b, f\}$, where pa, p, b and f stand for parrots, penguins, birds and fly respectively. The sentence $\bullet b \rightarrow f$ is equivalent to $b \mid\!\sim f$ in the KLM-Approach. But PTL allows for the extra expressibility of saying a sentence like

parrots are typical birds or $pa \to \bullet b$.
Entailment with PTL also makes use of Ranked Interpretation structures or Ranked Models. For the statement $\bullet\alpha$ to be considered satisfiable in the knowledge base, it must be true on the most typical level of α only in a ranked model. Entailment for PTL will be elaborated on in later sections of the paper.

3 ENTAILMENT FOR PTL

With the extra expressibility of PTL come challenges with the process of Entailment and Reasoning in PTL. In Booth et al [4], they show that the same Rational Closure Algorithm that is used in Defeasible Reasoning no longer works for PTL. Therefore adaptions need to be made for PTL for working out entailment.
Booth et al [3] [2] put forward ten postulates with regard to typicality which the consequence operator would need to satisfy in order for the rational closure algorithm to be able to be applied. The consequence operator is the set of all the consequences of a given knowledge base, which is basically the set of everything that can be reasoned from a given knowledge base. They show that, in the case of TL, it is impossible for all ten postulates to be satisfied simultaneously. This result is referred to as the Impossibility Result.

3.1 Impossibility Result

If we are given an entailment relation $\models\!\!\approx_?$ for PTL ($\mathcal{L}$), the associated consequence operator can be shown to be $\mathcal{K} \subseteq \mathcal{L}^\bullet$, $Cn_?(\mathcal{K}) := \{\alpha \in \mathcal{L}^\bullet \mid \mathcal{K} \models\!\!\approx_? \alpha\}$.[2] The consequence operator therefore is simply being defined as all the sentences (α) that entail from the knowledge base ($\mathcal{K}$) in the given language ($\mathcal{L}$).
With the Entailment Relation and the Consequence operator given above, the ten postulates put forward are as follows [3]:

P1 $\mathcal{K} \subseteq Cn_?(\mathcal{K})$ (Inclusion)
P2 If $\alpha \in Cn_?(\mathcal{K})$, then $Cn_?(\mathcal{K} \cup \{\alpha\}) = Cn_?(\mathcal{K})$ (Cumulativity)
P3 For every $\mathcal{K} \subseteq \mathcal{L}^\bullet$, $Cn_0(\mathcal{K}) \subseteq Cn_?(\mathcal{K})$, and for some $\mathcal{K}' \subseteq \mathcal{L}^\bullet$, $Cn_0(\mathcal{K}') \subset Cn_?(\mathcal{K}')$ (Ampliativeness)
P4 For some $\mathcal{K}, \mathcal{K}' \subseteq \mathcal{L}^\bullet$, $\mathcal{K} \subseteq \mathcal{K}'$ but $Cn_?(\mathcal{K}) \nsubseteq Cn_?(\mathcal{K}')$ (Defeasibility)
P5 $\mid\!\sim_{Cn_?(\mathcal{K})}$ is a rational conditional relation on $\mathcal{L}$(Conditional Rationality)
P6 For every $\mathcal{K} \subseteq \mathcal{L}^\bullet$, there is a ranked interpretation $\mathscr{R}$ s.t. for all $\alpha \in \mathcal{L}^\bullet$, $\alpha \in Cn_?(\mathcal{K})$ if and only if $\mathscr{R} \Vdash \alpha$ (Single Model)
P7 If $\mathcal{K}$ is a conditional knowledge base, then $\mid\!\sim_{Cn_?(\mathcal{K})} = \mid\!\sim^{rc}_{\mathcal{K}}$ (Extends Rational Closure)
P8 Let $\alpha \in \mathcal{L}$. Then $\alpha \in Cn_?(\mathcal{K})$ if and only if $\alpha \in Cn_0(\mathcal{K})$ (Strict Entailment)
P9 Let $\mathcal{K}$ be a conditional knowledge base and $\alpha \in \mathcal{L}$. Then $\alpha \in Cn_?(\mathcal{K})$ if and only if $\alpha \in Cn_0(\mathcal{K})$ (Conditional Strict Entailment)
P9′ Let $\mathcal{K} \subseteq \mathcal{L}$ and $\alpha \in \mathcal{L}$. Then $\alpha \in Cn_?(\mathcal{K})$ if and only if $\mathcal{K}$ entails α in classical propositional logic. (Classical Entailment)
P10 Let $\alpha \in \mathcal{L}$. Then $\bullet T \to \alpha \in Cn_?(\mathcal{K})$ if and only if $\bullet T \to \alpha \in Cn_0(\mathcal{K})$. (Typical Entailment)

In Booth et al [3] [2], these postulates are discussed and it is proven that they cannot all be satisfied simultaneously in PTL, producing the impossibility result. This means that more than one form of entailment is possible for PTL. Each form of Entailment can satisfy different subsets of the postulates. Two forms of Entailment have been suggested for PTL: LM and PT Entailment. Choosing between the two forms of entailment is in accordance with what postulates are needed to be satisfied in the specific situation.

3.2 PT Entailment

PT Entailment satisfies all the postulates except Conditional Rationality and Single Model. It is an an algorithm that focuses on making sure that the presumption of typicality (from Lehmann [10]) is maintained. It can produce more than one ranked model.
My project partner, Andrew Howe-Ely focuses on the elaboration of PT-Entailment and its implementation, while this paper focuses on LM Entailment and its implementation.

4 LM ENTAILMENT

LM Entailment focuses on extending the Rational Closure Construction and Algorithm from Conditional Knowledge Bases to Typical Knowledge Bases in PTL. It also focuses on keeping the property of having a Minimal Model as a main property. The Minimal Model created in LM-Entailment is referred to as the LM preference relation ($\trianglelefteq_{LM}$).
It is important to note that it must satisfy the Conditional Rationality (P5) and, therefore, the Single Model (P6) Postulate. It is also important to note that LM-Entailment also satisfies Conditional Strict Entailment (P9) and Classical Entailment (P9'). LM-Entailment actually satisfies all the postulates except Strict Entailment (P8).
The algorithm suggested in Booth et al [2] to create the LM-Preference relation is shown below. $\mathcal{U}$ represents the set of all interpretations, L_i represents the i-th level of the Ranked Model($\mathscr{R}$).

Step 1 Start with an initial Ranked Model where all the interpretations are on the first level: $L_0 = \mathcal{U}$
Step 2 Separate the interpretations which do not contradict the knowledge base with regards to the current ranked model from those that do. The set that does contradict is defined as C. This is shown as $C_i := [\![\mathcal{K}]\!]^{\mathscr{R}_i}$
Step 3 If the set of interpretations being moved is equivalent to the current level, return the current Ranked Model with the current level set to the infinite level: If $C_i = L_i$, return $\mathscr{R}_i$
Step 4 If not move all the interpretations in C_i up a level ($L_{i+1} = C_i$) in the Ranked Model (creating a new Ranked Model) and go to Step 2.

In Booth et al [2], it is shown that this algorithm always terminates, produces a ranked model and that the ranked model produced is the LM minimal model. Therefore this algorithm is all that is needed to produce the ranked model needed to perform entailment checks in LM Entailment.
With this Ranked Model, LM-entailment can be reduced to classical entailment checks which can be performed with a SAT-Solver. Therefore, this algorithm is crucial in the proof of concept of a LM Entailment Implementation.

5 LM IMPLEMENTATION

The LM Entailment Implementation [1] is the main focus of this paper. The implementation was done in Python. It falls into three major components: a SAT-Solver, Conversion and the Entailment Check Algorithm. Each of these components are discussed in the next few section.

5.1 SAT-Solver

SAT-Solvers are used to solve the Boolean Satisfiability Problem. The Boolean Satisfiability problem is a known NP-Complete Problem. Many Logic Problems, including entailment in PTL, can be reduced into a Boolean Satisfiability Problem.
The implementation of both LM and PT Entailment in this way transforms into building a wrapper around a SAT-Solver. Both my partner and I decided to use the Python implementation MINISAT, licensed by MIT, for our implementations.[2]. The implementation of the MiniSAT is called PyMiniSolvers.
The implementation of both entailment methods transforms into using the algorithm described in Booth et al [2] to create the ranked model(s).It then converts the relevant ranked models, knowledge bases and sentences into a format that is compatible with the SAT-Solver.
The SAT-Solver takes takes a 2D list in as input and produces True or False as an output. It takes in the atoms as numbers rather than letters. If the SAT-solver produces True, then the sentences and interpretations that have been passed into it are consistent with each other. If it produces False, then the sentences and interpretations that have been passed into it the SAT-Solver are inconsistent with each other.
A couple of examples of the format that is passed into SAT-Solver are as follows:

$$[[1, 2], [3, 4]]$$
$$[[-1], [2, 3]]$$

These statements represent $(a \lor b) \land (c \lor d)$ and $(\neg a) \land (b \lor c)$ respectively with each atom corresponding to a specific number.
Once everything has been converted into the format compatible with the SAT-Solver, there are three steps to use it successfully:

(1) Make unique variables for each of the atoms
(2) Add each of the sentences of the Knowledge Base as a clause
(3) Run the "solve" method and see if it produces true or false

In the case of an entailment check, there is one extra step added. The first two steps are the same. The inserted step would be to add the negation of the sentence being checked as a clause. The next step would be to run the "solve" method.
The negation of the sentence is taken, because the to solve the Boolean Satisfiability Problem for a set of sentences, the SAT-Solver would need to find an interpretation that satisfies the conjunction of all the sentences. If this is possible with the negation of the sentence being checked included in the set, it means that there is a direct contradiction to the sentence and, therefore, the sentence doesn't entail.Therefore, it follows that, in the case of entailment, if the SAT-solver returns true, then it is said that the sentence doesn't entail from the Knowledge Base. If the SAT-solver returns false, then it is said that the sentence does entail from the Knowledge Base.

[1]Implementation found at https://github.com/GuyAOrpGreen/PTR
[2]https://github.com/liffiton/PyMiniSolvers

5.2 Conversion

There are four steps that need to be taken in converting the problem into a form that can be used with the SAT Solver. The steps are as follows:

(1) Convert all sentences to Conjunctive Normal Form (CNF)
(2) Propagate the negation operators as far as possible in each sentence
(3) Propagate the or/disjunction operators as far as possible in each sentence.
(4) Convert each sentence into SAT-Solver Format (The 2D list with the atoms represented as numbers)

Converting to CNF affects all the sentences with an implication operator ($\rightarrow$) in them. The implication operator gets replaced with a disjunction/or operator ($\lor$) and the antecedent is negated. An example of this is shown in Equation 1. This can become tricky if there are nested implications.

$$a \rightarrow b \Rightarrow \neg a \lor b \tag{1}$$

Converting to CNF is implemented by looking at the bracket pairs in the sentence and assessing whether the operator in between them is an implication operator. It replaces the implication operator with a disjunction operator and adds a negation to the antecedent if it is. It starts with the outside brackets and once it has completed the replacement, it produces the new sentence without that implication. It then iterates this process until there is no longer an implication in the sentence. It also keeps track of indices changes of the brackets, as adding negation operators increases the indices of some of the other brackets.
Moving onto the second step: propagating the negation operators within the furthermost inside bracket. To propagate the negation, De Morgan's Laws are followed. If you are distributing negation into a bracket, negation is taken of the antecedent and the consequence inside the bracket, as well as the corresponding operator. Each or/disjunction operator ($\lor$) is replaced with and and/conjunction operator ($\land$) and vice versa. If there are two negations in row ($\neg\neg$), they cancel each other out. An example of distributing negations is shown below:

$$\neg(a \land b) \Rightarrow (\neg a \lor \neg b) \tag{2}$$

In the implementation, this is done recursively. The first step in each recursive call to remove any double negations. From there it then determine where the next negation that needs to be propagated is in the sentence by looking for the first occurrence of "¬(". It then distributes that negation into the next inner brackets.It does this by applying De Morgan's Laws (described above). It limits this application to only the current bracket pair, ignoring other bracket pairs that may be inside it. This process produces a new sentence. The function is then called again with the new sentence. Intuitively this means that it works systematically from the outer brackets inwards on each side of the main operator. The base case is if the string doesn't contain '¬('. This would mean that there is not any negation operator that would need to be propagated.
Propagating the or/disjunction operators ($\lor$) is needed to make

it easier to convert the sentence to SAT-solver format. This ensures that the and/conjunction operators ($\wedge$) are referring to the outermost brackets. As soon as there is an or/disjunction operator, any inner brackets from that point can only have disjunction/or operators and no and/disjunction operators. This can be ensured by manipulating the disjunction and conjunction operators within these brackets. An example of this is shown below:

$$(a) \vee (b \wedge c) \Rightarrow (a \vee b) \wedge (a \vee c) \tag{3}$$

This algorithm is done recursively. It goes through each bracket pair from the outermost to the innermost and as soon as there is a disjunction/or operator, it ensures that there are no more conjunction/and operators. If there aren't, it returns the sentence. If there are, it retrieves the first occurrence of a conjunction/and operator and fixes it. It does this by first determining if the conjunction/and operator is in the antecedent or consequence of the relevant disjunction/or operator. It then replaces the disjunction/or operator with a conjunction/and operator. It takes the antecedent and consequence of the violating *conjunction/and operator* and takes the conjunction of each of the antecedent and consequence taken with the disjunction of the antecedent of the original *disjunction/or* operator that got replaced earlier if the violating conjunction/and operator occurs in the *consequence.* It takes the antecedent and consequence of the violating *conjunction/and operator* and takes the conjunction of each of the antecedent and consequence taken with the disjunction of the consequence of the original *disjunction operator* that got replaced earlier if the violating conjunction/and operator occurs in the *antecedent* of that operator. The function is then called again with the new sentence.
To show an example of how the propagating disjunction algorithm works: Given this sentence where α, γ, ω and β can be any sentence

$$((\gamma) \wedge (\omega)) \vee ((\alpha) \vee (\beta))$$

The implementation checks whether or not propagation of the disjunction operators are needed. It is seen in this case that it is needed as the outermost brackets are separated by a disjunction/or operator and the inner brackets on the left (the antecedent) are separated by a conjunction/and operator. It then looks at which of the antecedent or consequence of the *disjunction/or operator* is the violation occurring.In this case it is the antecedent as it is on the left hand side. So it then takes the antecedent $((\gamma))$ and consequence $((\omega))$ of the relevant*conjunction/and operator* and creates the conjunction of each of them with the disjunction of the consequence of the *disjunction/or operator* $(((\alpha) \vee (\beta)))$. And so it produces the sentence:

$$((\gamma) \vee ((\alpha) \vee (\beta))) \wedge ((\omega) \vee ((\alpha) \vee (\beta)))$$

It then calls the function again with this new sentence. Dependent on the sentences within the brackets, it will determine whether propagation of disjunction is needed or not.
The last step is converting to SAT-Solver Format, which requires the conversion of each sentence into a 2D list in which each atom is represented by a unique number, negation is represented by a '-' and disjunction/or operators are represented by lists and conjunction/and operators are represented by making a new list. An example of this is shown below:

$$(a \vee b) \wedge (\neg b \vee c) \Rightarrow [[1, 2], [-2, 3]] \tag{4}$$

This is done by going through the string from left to right. At the start a list is made. If the index being looked at is a negation, a boolean is set to True. If the index being looked at is an atom, the corresponding number is appended to the "other" list with a minus if the boolean is true. The boolean is reset to False afterwards. If the index is an or/disjunction operator, nothing is done. If the index is an and/conjunction operator. The current "other" list is appended to the list that was made at the start, and a new "other" list is made. If all the steps in the conversion described above are done correctly, this method will convert each sentence correctly into SAT-Solver Format

5.3 Entailment Check Algorithm

This is the first and only part of the implementation that considers the Typicality Operator(●). The rest of the sections ignore that the typicality operator exists.
The creation of the Ranked model used to perform LM Entailment is fairly similar to the Rational Closure Construction and Algorithm. The algorithm was first described in Booth et al [2]. The algorithm involves looking at all the possible interpretations from a given knowledge base and moving them up levels of the ranked model, depending on their typicality. This can be quite computationally expensive as there are 2^n interpretations for n atoms.
The first step in the implemented algorithm is to create all the possible interpretations. Then these interpretations are all added to the first level of the ranked model. Once that is complete, each interpretation is looked at individually with the Knowledge Base. Each interpretation is either left on its current level, if it is consistent with the knowledge base, or it is moved up one level if it's inconsistent. The typicality operator can be ignored in the consistency check of each interpretation with the knowledge whilst all the interpretations are on the first level. This ignoring of the typicality operator can only be done for moving up interpretations from the first level of the knowledge base and it can only be done because nothing will be non-typical on the first level.
For computing the rest of the levels, the implementation runs a while loop first checking which atoms are not typical on that current level. It does this by checking all the previous levels of the ranked model and returning all the atoms that have been true in at least one interpretation in any of the lower levels. It then systematically searches every sentence in the knowledge base for the typicality operator.. For each sentence that contains the typicality operator, the atoms (for which the typicality operators apply to) are checked whether they are typical or not on the level using what has already been worked out earlier. If the atom being checked is typical, the typicality operator that refers to that atom is ignored for the specific level. If the atom being checked is not typical, the atom is made to be unconditionally false for each sentence which contains the typicality operator applying to that atom. For example if p is not typical on the current level and we have the sentence $\bullet p$, it gets replaced by $p \wedge \neg p$. $p \wedge \neg p$ is unconditionally false. This is done for all the sentences in the knowledge base that contain typicality operator(s) and all the relevant atoms. Once this is completed, all the interpretations are checked individually with the new knowledge base to see if they are consistent.
If an interpretation is not consistent, it is moved up a level. If it is

consistent, it's left on the current level. This is repeated until the level below the current level in the loop is empty. Any interpretations that are remaining are put on the "infinity" level (L_∞) and disregarded. From there, we have a ranked model and all that needs to be checked is entailment.
To check if a classical sentence entails from the ranked model, the sentence is checked against each relevant interpretation in the ranked model (besides L_∞). The sentence will have to be consistent with every relevant interpretation in the ranked model. This check is done by passing in the negation of the sentence and the interpretation to the SAT-Solver. If it produces False, then it is consistent and if it produces True then it doesn't.
To check if a sentence that contains typicality entails from the ranked model, a similar process is followed. The implementation first returns for each level what atoms are most typical on that level. The next step in the process is to check which atoms the typicality applies to in the sentence being checked. Then the same process as above for a classical sentence is followed, except only for the relevant interpretations on the level for which the atoms are most typical.

5.4 Testing

Simple knowledge bases were tested with the implementation. They were either compared with the theoretical results the algorithm should produce, or with Andrew's implementation (which should also produce the same ranked model as one of its models).
In the case of comparing with Andrew's ranked model, the ranked models produced were compared and if PT Entailment only produced one ranked model, entailment of specific intuitive sentences were compared as well. PT Entailment does take longer computationally than LM Entailment and so larger knowledge Knowledge Bases weren't able to be compared.
The implementation was also tested to see if it could reproduce the LM-Minimal Model result for the example Knowledge Base shown in Booth et al [2]. The implementation successfully replicated the LM-Minimal Model that was theoretically produced.
A couple of examples of knowledge bases and the ranked models produced in LM Entailment are shown below.

5.4.1 *Example 1: Tweety Example.*

The Knowledge Base for the *Tweety* Example in PTL is as follows: $\mathcal{K}$ =["$p \rightarrow b$", "$\bullet b \rightarrow f$", "$\bullet p \rightarrow \neg f$"]. Where p, b and f stand for Penguins, birds and flying respectively.
The LM-Minimal model produced is shown in Figure 2. The first value represents the boolean value for penguin. The second value represents the boolean value for bird and the third value represents the boolean value for flying. This ranked model is the same ranked

L_∞ :	100		101
L_2 :		111	
L_1 :	010		110
L_0 :	000	001	011

Figure 2: The LM-Minimal Model for the *tweety* example

model that is produced in the KLM-Approach. This is correct as the sentence $\bullet b \rightarrow f$ is equivalent to $b \mid\sim f$ and similarly with the sentence $\bullet p \rightarrow \neg f$ and $p \mid\sim \neg f$.
From this Ranked Model, the implementation was able to entail that $p \rightarrow \neg \bullet b$ or that *penguins are not typical birds.* This corresponds to the intuition of the problem.

5.4.2 *Example 2: Flying Fish Example.*

The Knowledge Base for the *Flying Fish* Example in PTL is as follows: $\mathcal{K}$ =["$\bullet b \rightarrow f$", "$e \rightarrow \neg \bullet f$", "$e \rightarrow \neg b$", "$\bullet f \rightarrow w$"]. Where b, f, e and w stand for birds, flying, flying fish (Exocoetidae is the scientific name) and wings respectively.
The LM-Minimal model produced is shown in Figure 3. The first value represents the boolean value for bird. The second value represents the boolean value for flying. The third value represents the boolean value for flying fish and the fourth value represents the boolean value for wings. From this model, if you try to entail if

L_∞ :		1010	1011	1110	1111	
L_1 :	0100	0110	0111	1000	1001	1100
L_0 :	0000	0001	0010	0011	0101	1101

Figure 3: The LM-Minimal Model for the *Flying Fish* example

flying fish have wings, it produces false as the answer which on the face of if would make sense, as the knowledge base says that Flying Fish are not typical flying things and typical flying things have wings. In this case it is not inferring whether or not Flying Fish have wings as it does not mention what non-typical flying things may have.

5.5 Format

In the implementation, atoms can only be single letters. The operators are represented as follows:

- Disjunction ($\vee$): "|"
- Conjunction ($\wedge$): "&"
- Implication ($\rightarrow$): ">"
- Negation ($\neg$): "$-$"
- Typicality ($\bullet$): "$*$"

Sentences are represented as strings and knowledge bases are represented by lists of these strings.
The implementation takes knowledge bases and test sentences in this form and would need to convert it into a format the SAT solver would understand.
The sentences would need pairs of bracket around the relevant antecedent and the consequence for each operator to work properly as well.

5.6 Limitations

The implementation does have many limitations. They include:

(1) Atoms can only be a single letter
(2) The typicality operator can only apply to one letter
(3) The conversion not completely accurate for very large sentences
(4) Sentences need extra brackets that would not be necessary for understanding to do the conversion correctly

(5) There is no equivalence operator coded in the implementation

For some of these there is a way around it still using the implementation. For example, for the equivalence operator, $a \leftrightarrow b$ is exactly the same as $(a \rightarrow b) \wedge (b \rightarrow a)$, which the implementation can handle.

A possible solution to the typicality operator only applying to one letter is to create a new letter (or atom) which is defined as the equivalent of your need. The typicality operator can then be applied to this new letter.The solution discussed above would need to be used to be able to represent the equivalence operator.

Limitation in terms of testing also exist: the bigger the knowledge bases get, the more difficult it becomes to create theoretical answers to compare to what the algorithm creates and the more computationally expensive it becomes to run the implementation. However, this is beyond the scope of the paper as this is a proof of concept of an implementation of LM Entailment.

6 CONCLUSIONS

To conclude, it has been shown that it is possible to reduce an Entailment check in PTL to a series of Classical Entailment Checks for LM-Entailment. These classical entailment checks can then be effected by a SAT-Solver of choice.

A proof of concept of LM Entailment in PTL has also been shown. However, difficulties do occur with the conversion to the SAT-Solver format and, once having produced the ranked model, producing sample Knowledge Bases and sentences that are able to test all of the properties of entailment in PTL also becomes an increasingly difficult problem.

There are also limitations of the current implementation. One of the consequences of the limitations of the current implementation is that there can only be twenty six different atoms as atoms can only be single letters. This limits this implementation in the sense that large knowledge bases with more that twenty six atoms are not able to be tested on this implementation. It also limits the use of the solution proposed for the limitation of the typicality operator applying to only one atom. This is because each time the solution is used, a new atom needs to be created.

It can be seen that LM Entailment satisfies all of the suggested postulates except Strict Entailment. LM Entailment has also been shown to be less computationally expensive than PT Entailment, but it does limit some of the conclusions that would want to be made with PTL.

Whether to use LM or PT Entailment in PTL would be determined by the context of the problem.

6.1 Future Work

Obvious future work can be related to addressing each of the five limitations of the implementation mentioned above. With the main limitation being that atoms can only be single letters.

Future work can also include exclusively working on creating Knowledge Bases and theoretical results that test all of the properties of entailment in PTL.

The computational complexity in terms of space and time of the implementation was not considered at all in this proof of concept, so a possible extension is to make the implementation more efficient.

7 ACKNOWLEDGEMENTS

I would like to thank everyone who was prepared to proof-read my project and give me feedback.

I would like to thank my project partner, Andrew Howe-Ely, for all the help he has given me and for providing me with his results so I could make a comparison between the two studies.

I would also like to thank Professor Tommie Meyer for the advice, support, encouragement and for everything he taught me to make this project possible.

REFERENCES

[1] Mordechai Ben-Ari. 2012. *Mathematical logic for computer science* (3 ed.). Springer Science & Business Media.

[2] Richard Booth, Giovanni Casini, Thomas Meyer, and Ivan Varzinczak. 2015. On the Entailment Problem for a Logic of Typicality. In *Proceedings of the 24th International Joint Conference on Artificial Intelligence (IJCAI)*. 2805–2811.

[3] Richard Booth, Giovanni Casini, Thomas Meyer, and Ivan Varzinczak. 2015. What Does Entailment for PTL Mean?. In *Proceedings of the 12th International Symposium on Logical Formalizations of Commonsense Reasoning.*

[4] Richard Booth, Thomas Meyer, and Ivan Varzinczak. 2012. PTL: A Propositional Typicality Logic. In *Proceedings of the 13th European Conference on Logics in Artificial Intelligence (JELIA) (LNCS)*, L. Fariñas del Cerro, A. Herzig, and J. Mengin (Eds.). Springer, 107–119.

[5] Richard Booth and Jeff B Paris. 1998. A Note on the Rational Closure of Knowledge Bases with Both Positive and Negative Knowledge. *Journal of Logic, Language and Information* 7, 2 (1998), 165–190.

[6] SA Cook. 1971. The complexity of theorem-proving procedures. In *Proceedings of the third annual ACM symposium on Theory of computing*. ACM, 151–158.

[7] Laura Giordano, Valentina Gliozzi, Nicola Olivetti, and Gian Luca Pozzato. 2012. A minimal model semantics for nonmonotonic reasoning. In *Logics in Artificial Intelligence*. Springer, 228–241.

[8] W Gong and X Zhou. 2017. A survey of SAT solver. In *AIP Conference Proceedings*, Vol. 1836. AIP Publishing, 020059.

[9] Sarit Kraus, Daniel Lehmann, and Menachem Magidor. 1990. Nonmonotonic reasoning, preferential models and cumulative logics. *Artificial Intelligence* 44 (1990), 167–207.

[10] Daniel Lehmann. 1995. Another perspective on default reasoning. *Annals of Mathematics and Artificial Intelligence* 15, 1 (1995), 61–82.

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