Propositional Typicality Reasoning using PT Entailment

Andrew Howe-Ely
University of Cape Town
hwland004@myuct.ac.za

ABSTRACT

Propositional Typicality Logic (PTL) is a logic that allows for exceptions to be made in the name of knowledge representation. It is an extension of classical propositional logic with the addition of a typicality operator. This paper provides the background of reasoning in PTL using entailment, and what this should seek to achieve. Specifically one such proposed form is explored: PT Entailment. It presents an implementation of PT Entailment as a proof of concept. Testing of this implementation found it to be correct and produce the correct minimal models and entailment results. However, it proved inefficient, especially for large or complex knowledge bases, with the algorithm having exponential complexity.

KEYWORDS

Defeasible Reasoning, Entailment, Non-monotonicity, Knowledge Representation, Propositional Logic

1 INTRODUCTION

Propositional Typicality Logic (PTL) is a logic proposed by Booth et al. [3] that includes the notion of typicality. The language of PTL is that of classical propositional logic with the extension of the typicality operator, $\bullet$, which captures the most normal situations in which a statement holds. This paper presents a description of an algorithm for reasoning in PTL and its implementation, and the necessary background. Reasoning in this setting amounts to computing entailment. Two forms of entailment have been defined for PTL [2]: one of these, PT Entailment, is presented in this paper.

While PTL is an extension of classical propositional logic, entailment in it is based on the work in defeasible reasoning. Defeasible statements are those of the form "typically if a, then b". Reasoning in a defeasible setting is non-monotonic, meaning conclusions can be retracted if more information becomes known. A knowledge base refers to a set of statements from which conclusions can be drawn. Defeasible reasoning is done using Rational Closure, which is also the basis of entailment in PTL. The language of PTL allows for the creation of more expressive statements containing typicality. Defeasible conditionals can only contain typicality in the antecedent of a rule, whereas the $\bullet$ can be placed anywhere in a statement in the language of PTL.

The motivation for PTL, and indeed defeasibility, is to express the exceptions in a set of rules. This is easily done for humans in the real world, as we can easily handle exceptions to general rules, but becomes more complex to represent in logic. For example, we know that while most birds fly, penguins are an exception. The basis of classical propositional logic would not allow any exceptions, as these would bring up contradictions. To be able to express exceptionality is therefore an interesting problem in the knowledge representation and reasoning field of AI.

This paper describes an implementation of a reasoner for PTL using PT Entailment by producing a wrapper around an existing SAT Solver. A SAT Solver is an implementation of an algorithm to solve the Boolean Satisfiability Problem, an NP-Complete problem. This can be used to compute classical entailment and check satisfaction of sentences in classical propositional logic. My project partner, Guy Green, focused on the implementation of the other proposed form of entailment, LM Entailment.

The main aim of this project is to show that PT Entailment could be implemented computationally and produce the same results as those derived theoretically. This serves as a proof of concept of the correctness of the algorithm. This paper will first give the necessary background for classical and defeasible reasoning using propositional logic in Section 2. Section 3 will provide the background of PTL and define 11 properties that the authors Booth et al. put forward as appropriate for entailment in PTL, as well as the result that they cannot all be satisfied at once. With this in mind, PT Entailment is described along with the algorithm used to reason in it. Section 6 describes the computational implementation of this algorithm in the form of a program coded in Python and the results of testing it using small knowledge bases.

2 BACKGROUND

In order to understand PTL, a description of the background work is required. The starting point is propositional logic, since PTL is an enriched version of classical propositional logic. Section 2.2 will provide the background of defeasible reasoning.

2.1 Propositional Logic

Propositional logic is a logical system that builds up sentences from propositional atoms and logic operators. Each atom represents a value that is either true or false. Atoms are represented with letters such as p and q. The connectives between atoms are the following logic operators: And/Conjunction: $\wedge$, Or/Disjunction: $\vee$, Not/Negation: $\neg$, Implication: $\rightarrow$ and Equivalence: $\leftrightarrow$. These are all binary operators, besides negation. $\top$ and $\bot$ mean unconditionally true or false respectively. All of the other connectives and symbols can be represented as a combination of negation, conjunction and disjunction. This fact is important since in the implementation of a propositional system everything will be broken down into those terms.

A sentence built up from these atoms and connectives is either entirely true or false. Let $\mathcal{P}$ be a set of propositional atoms, the language $\mathcal{L}$ from $\mathcal{P}$ is built up of sentences α, β. Here α is statement built from atoms such as p and q connected by the classical operators.

An interpretation is what gives meanings to the symbols in a sentence, it is a function which assigns a value of true or false to every atom in a sentence. Each interpretation can also be referred to as a possible world. For a sentence containing n atoms there are 2^n

interpretations. A valuation is another term for this same concept, and is defined as a function $v : \mathcal{P} \longrightarrow \{0, 1\}$. Let $\mathcal{U}$ be the set of all possible valuations for $\mathcal{P}$. These valuations can be represented as binary strings or as a set of literals, e.g. if $\mathcal{P} = \{p, q\}$, the valuation where p is true and q is false can be written as 10 or as $\{p, \neg q\}$.

A sentence is called *satisfiable* if there exists an interpretation in which it evaluates as true. Notationally this is shown as: $v \Vdash A$, where v is an interpretation that satisfies a sentence A. A satisfying interpretation is called a model.

From this, logical consequences can be defined. A sentence A is a logical consequence of a set of sentences U if and only if every model of U is also a model of A. This is denoted by $U \models A$, and can also be thought of as $Mod(U) \subseteq Mod(A)$, where $Mod(A)$ denotes the models of A. This is the form of entailment used in a classical setting. This form of entailment is monotonic. If we add a statement to a knowledge base, anything entailed by the original knowledge base will still be entailed.

To show how one can model the real world in propositional logic, we take an example that will be referred back to later. The classical Tweety example says that $p \rightarrow b$, $b \rightarrow f$. This can be read as: *penguins are birds* and *birds fly*. From this we would reason that penguins fly, or $p \rightarrow f$. If we added the sentence $p \rightarrow \neg f$, meaning *penguins do not fly*, we would reason that there are simply no penguins. $\neg p$ would be a consequence, as well as $p \rightarrow f$. This is a demonstration of monotonicity, since more information was added but the previous consequences still held. Clearly, this is not what we desire in a situation where we want to handle exceptionality.

2.2 Defeasible Reasoning

Defeasible reasoning extends reasoning in propositional logic by being able to deal with exceptions to rules. This allows for the creation of statements that were not previously able to make in classical propositional logic. Recall the Tweety example from propositional logic, where given an exception that penguins do not fly lead us to conclude there are no penguins. With defeasibility we say that *birds typically fly*, and then conclude that penguins are simply exceptional birds that do not fly. In order to do this a new connective is introduced: $\mid\!\sim$. This is a new type of conditional. The statement $a \mid\!\sim b$ reads as: if a, then typically b. This approach is referred to as the KLM approach and was investigated in Kraus et al. [7]

Defeasible reasoning is non-monotonic, meaning when new information is added that conflicts with previous conclusions, these conclusions can be withdrawn. If this were not the case one would not be able to make the exceptions needed to express something as typical.

The $\mid\!\sim$ operator is said to be a preferential conditional since it has the following properties, put forward by the authors Kraus et al.: Reflexivity (Ref), Left Logical Equivalence (LLE), Right Weakening (RW) and Cautious Monotonicity (CM).

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

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

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

Another property that $\mid\!\sim$ has is Rational Monotonicity (RM). From this we can say it is a rational conditional.

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

In Lehmann and Magidor [8] ordered structures called ranked interpretations are used to reason with the defeasible conditional. A ranked interpretation is an ordering of different interpretations in the set $\mathcal{V}$ using the ordering $\prec$, but can be written as a partition $\mathscr{R} = (L_0, \ldots, L_n)$. The valuations lower down in the ordering are the ones that are more typical. Valuations that are not in the ranked model are defined as being on L_∞. We say a ranked interpretation is a ranked model of some set of conditionals if it entails every conditional in that set. A new preference relation $\trianglelefteq_{\text{LM}}$ is defined in Lehmann and Magidor's work. This forms a partial order over ranked interpretations. The intuition is that the ranked interpretation with the most typical valuations possible at the bottom of the ranking should be preferred. Using this order, there is shown to be a unique minimal ranked interpretation among all the ranked models. This work is derived from Lehmann and Magidor and further developed by Booth and Paris [4] and Giordano et al. [5]. Reasoning in this KLM framework is thought of as deriving new conditionals from a set of conditionals C. The Rational Closure of the set C is $\mid\!\sim^{\text{rc}}_C := \{(\alpha, \beta) \mid \mathscr{R}^{\text{rc}}(C) \Vdash \alpha \mid\!\sim \beta\}$. The rational closure construction gives us a form of entailment for defeasibility. This is not the only form of entailment for defeasibility, but it is the one which entailment in PTL is based on. A second method of entailment is the rational closure algorithm, which reduces entailment of defeasible conditionals to a series of classical entailment checks.

Figure 1 shows a ranked interpretation for the knowledge base $\mathcal{K} = \{b \mid\!\sim f, p \rightarrow b, p \mid\!\sim \neg f\}$ when using rational closure. Let's interpret these as *birds typically fly, penguins are birds, penguins typically don't fly*. This ranked interpretation is a model for $\mathcal{K}$, meaning it satisfies it.

L_2 :		111	
L_1 :	100		101
L_0 :	000	010	110

Figure 1: A ranked interpretation for $\mathcal{P} = \{b, f, p\}$.

2.3 SAT

SAT refers to the Boolean Satisfiability Problem [9]. This problem is to determine whether given a formula, there exists an interpretation

that satisfies it. This problem is NP-Complete, and there is no algorithm that is known to solve each SAT problem. However, there are SAT-algorithms that can solve SAT problems sufficiently for most practical uses. In many cases, in order to solve SAT problems, the formulas need to be in a specific format: Conjunctive Normal Form (CNF). This form is the conjunction of several clauses of literals. A literal is either a variable, or the negation of that variable. Each clause is the disjunction of literals. For example, a clause would be $x \vee \neg y$. In this sense, the sentences must be an *and* of *or*s. Since every other Boolean connective can be represented as a combination of $\vee$, $\wedge$ and $\neg$, it is possible to convert every propositional sentence into CNF. However, sometimes this means the sentences will exponentially grow. An example below shows the conversion of such a sentence into CNF:

$$a \vee (b \wedge c) \longrightarrow (a \vee b) \wedge (a \vee c) \tag{1}$$

A formula in CNF can be checked for satisfiability by a SAT-solver, an implementation of an algorithm that solves SAT. Several implementations exist [6] and one was chosen for the implementation in Section 6.

3 PROPOSITIONAL TYPICALITY LOGIC

Propositional Typicality Logic (PTL) is an extension to propositional logic by adding the typicality operator, $\bullet$. It was introduced by Booth et al [3] [2]. This operator captures the worlds that are most typical. This means the sentence $\bullet\alpha$, meaning the situations in which α is most typical, can now be part of the language. The typicality operator can be put anywhere in the sentence, and α itself can be a typicality sentence. The language of PTL sentences is $\mathcal{L}^{\bullet}$. The sentence $\bullet b \rightarrow f$ can be read as *typical birds fly*. This has the same meaning semantically as $b \mid\sim f$, and will generate similar results. However, the power of PTL is that the typicality can be placed anywhere, not just in the antecedent of a rule. For example, $\bullet b \rightarrow \bullet f$ is possible, and would read as *typical birds are typical flying things*. This added expressivity allows for more varied sentences which contain information about typicality.

Ranked interpretations are also used for the semantics of PTL. Satisfaction of a sentence $\bullet\alpha$ by v in $\mathscr{R}$ is defined as $v \Vdash \alpha$ and there is no v' lower than v in $\mathscr{R}$ such that $v' \Vdash \alpha$. In other words, v must be the most typical world in which α holds in order to satisfy $\bullet\alpha$. A ranked model of α is a ranked interpretation that satisfies α. A PTL knowledge base, $\mathcal{K}$, is a finite set of PTL sentences in $\mathcal{L}^{\bullet}$. The models, $Mod(\mathcal{K})$, of a knowledge base are the ranked interpretations that satisfy the conjunction of all sentences in it.

3.1 Entailment in PTL

Entailment in PTL should also be defeasible. In fact there are a number of postulates suggested by Booth et al. that a form of entailment for PTL could satisfy. However, they show the result that these cannot all be simultaneously satisfied. This impossibility result means that multiple forms of entailment are possible for PTL.

The authors define ranked entailment for a typicality knowledge base, as $\mathcal{K} \mid\approx_0 \alpha$ if and only if $Mod(\mathcal{K}) \subseteq Mod(\alpha)$. They show this form of entailment to be unsuitable for PTL [2]. Ranked entailment forms the underlying form of entailment for entailment in typicality. For a typicality entailment relation $\mid\approx_{\bullet}$, its consequence operator $Cn_{\bullet}$ is the set of consequences of a knowledge base. It is defined as $Cn_{\bullet} := \{\alpha \in \mathcal{L}^{\bullet} \mid \mathcal{K} \mid\approx_{\bullet} \alpha\}$. The consequence operator of ranked entailment is defined similarly, and denoted Cn_0.

The first two properties are:

P1 $\mathcal{K} \subseteq Cn_{\bullet}(\mathcal{K})$ (Inclusion)

P2 If $\alpha \in Cn_{\bullet}(\mathcal{K})$, then $Cn_{\bullet}(\mathcal{K} \cup \{\alpha\}) = Cn_{\bullet}(\mathcal{K})$ (Cumulativity)

These two properties are satisfied by the consequence operators of ranked entailment.

The following two properties satisfy the need for entailment in PTL to be non-monotonic.

P3 For every $\mathcal{K} \subseteq \mathcal{L}^{\bullet}$, $Cn_0(\mathcal{K}) \subseteq Cn_{\bullet}(\mathcal{K})$, and for some $\mathcal{K}' \subseteq \mathcal{L}^{\bullet}$, $Cn_0(\mathcal{K}') \subset Cn_{\bullet}(\mathcal{K}')$ (Ampliativeness)

P4 For some $\mathcal{K}, \mathcal{K}' \subseteq \mathcal{L}^{\bullet}$, $\mathcal{K} \subseteq \mathcal{K}'$ but $Cn_{\bullet}(\mathcal{K}) \nsubseteq Cn_{\bullet}(\mathcal{K}')$ (Defeasibility)

Ampliativeness says that this form of entailment should give at least as many consequences as the classical form of entailment. While defeasibility is the condition that previously drawn conclusions can be withdrawn upon the addition of more information.

The next property is that the defeasible conditional induced by $Cn_{\bullet}(\mathcal{K})$ should be a rational conditional, as defined previously, by meeting all the rationality properties.

P5 $\mid\sim_{Cn_{\bullet}(\mathcal{K})}$ is a rational conditional relation on $\mathcal{L}$(Conditional Rationality)

P5 would also imply P4. The next property is a strengthening of P4.

P6 For every $\mathcal{K} \subseteq \mathcal{L}^{\bullet}$, there is a ranked interpretation $\mathscr{R}$ s.t. for all $\alpha \in \mathcal{L}^{\bullet}$, $\alpha \in Cn_{\bullet}(\mathcal{K})$ if and only if $\mathscr{R} \Vdash \alpha$ (Single Model)

When $\mathcal{K}$ is a conditional knowledge base, made up of statements of the form $\bullet\alpha \rightarrow \beta$, the result should coincide with that of rational closure.

P7 If $\mathcal{K}$ is a conditional knowledge base, then $\mid\sim_{Cn_{\bullet}(\mathcal{K})} = \mid\sim^{\mathrm{rc}}_{\mathcal{K}}$ (Extends Rational Closure)

The next property was shown to be satisfied by rational closure.

P8 Let $\alpha \in \mathcal{L}$. Then $\alpha \in Cn_{\bullet}(\mathcal{K})$ if and only if $\alpha \in Cn_0(\mathcal{K})$ (Strict Entailment)

This states that $Cn_{\bullet}$ should produce the same results as ranked entailment when the sentences do not contain the typicality operator.

P9 Let $\mathcal{K}$ be a conditional knowledge base and $\alpha \in \mathcal{L}$. Then $\alpha \in Cn_{\bullet}(\mathcal{K})$ if and only if $\alpha \in Cn_0(\mathcal{K})$ (Conditional Strict Entailment)

This is a weaker version of strict entailment that only requires it to hold for conditional knowledge bases.

P9′ Let $\mathcal{K} \subseteq \mathcal{L}$ and $\alpha \in \mathcal{L}$. Then $\alpha \in Cn_{\bullet}(\mathcal{K})$ if and only if $\mathcal{K}$ entails α in classical propositional logic. (Classical Entailment)

This requires the form of entailment to coincide with classical propositional entailment.

P10 Let $\alpha \in \mathcal{L}$. Then $\bullet\top \rightarrow \alpha \in Cn_{\bullet}(\mathcal{K})$ if and only if $\bullet\top \rightarrow \alpha \in Cn_0(\mathcal{K})$ (Typical Entailment)

This last property is similar to P8.

Booth et al. then provide the result that these cannot all be satisfied at once.

Theorem 3.1. *There is no PTL consequence operator $Cn_\bullet$ satisfying all of P1, P6, P8 and P10.*

This means that more than one form of entailment is possible for PTL, dropping certain properties. The authors point to this as a result of the expressive nature of PTL. Two forms of entailment are put forward, LM and PT Entailment. The latter being the focus of this project and will be discussed more in the next section.

4 PT ENTAILMENT

PT Entailment is a form of entailment for PTL that satisfies Strict Entailment, but not Conditional Rationality and therefore not the Single Model postulate.

PT Entailment is based on the minimality of ranked models. The algorithm to find minimal models can return multiple models. The idea behind this is to preserve the presumption of typicality, which corresponds to considering only models where valuations are as far down as possible in the ranking, meaning more typical. To compare ranked models a preference relation $\trianglelefteq_{\text{PT}}$ is introduced using a function defined as the height function. The height of a valuation in a ranked model is the number of the layer it is on, or if it is not in the ranked model then the height is infinity.

Definition 4.1 (Height Function). *For a ranked interpretation $\mathscr{R} = (L_1, ..., L_n)$ and valuation $v \in \mathcal{V}$, $h_{\mathscr{R}}(v) = i$ if $0 \leq i \leq n$ or ∞ otherwise.*

Using this height function the relation $\trianglelefteq_{\text{PT}}$ is defined. This relation is a preorder. When comparing two ranked interpretations, $\mathscr{R}_1$ and $\mathscr{R}_2$, $\mathscr{R}_1 \trianglelefteq_{\text{PT}} \mathscr{R}_2$ if and only if the height of every valuation in $\mathscr{R}_1$ is lower than its height in $\mathscr{R}_2$.

Definition 4.2 (Relation $\trianglelefteq_{\text{PT}}$). *For two ranked interpretations $\mathscr{R}_1$ and $\mathscr{R}_2$, $\mathscr{R}_1 \trianglelefteq_{\text{PT}} \mathscr{R}_2$ if and only if $h_{\mathscr{R}_1}(v) \leq h_{\mathscr{R}_2}(v)$ for every $v \in \mathcal{U}$. $\mathscr{R}_1 \triangleleft_{\text{PT}} \mathscr{R}_2$ if and only if $\mathscr{R}_1 \trianglelefteq_{\text{PT}} \mathscr{R}_2$ and not $\mathscr{R}_2 \trianglelefteq_{\text{PT}} \mathscr{R}_1$.*

This relation is important as it tells us which models are more preferred, with the PT-minimal models being the minimal models according to this relation. The minimal models of a knowledge base $\mathcal{K}$ is denoted by $min_{\trianglelefteq_{\text{PT}}}(Mod(\mathcal{K}))$. PT Entailment is formally defined as:

Definition 4.3. *$\mathcal{K} \models_{\trianglelefteq_{PT}} \alpha$ if and only if $min_{\trianglelefteq_{\text{PT}}}(Mod(\mathcal{K})) \subseteq Mod(\alpha)$*

To put this in words, it says a knowledge base entails a sentence α if and only if every valuation in the PT-minimal models of $\mathcal{K}$ is also a model for α. The consequence operator for PT Entailment is Cn_{PT}. Cn_{PT} satisfies P1-4, P7 and P8-10.

4.1 LM Entailment

In comparison, LM Entailment satisfies every postulate except for P8; Strict Entailment. The algorithm for LM Entailment returns a single minimal model built by moving valuations up in the ranked model if they do not satisfy a given knowledge base with respect to the ranked model. This algorithm terminates when valuations being moved up a level are the same as the previous iteration. This is in contrast to PT Entailment, where multiple minimal models are returned and valuations are not moved up the ranked models but different models are tested.

5 PT ENTAILMENT ALGORITHM

PT Entailment requires us at look at different ranked interpretations to determine minimal models. Here the algorithm for determining these models will be presented.

The algorithm will return a set of PT-minimal models $\mathfrak{R}_{\mathcal{K}}$. To do this systematically all possible interpretations of the atoms $\mathcal{P}$ in a knowledge base $\mathcal{K}$ must be considered. First, all valuations which do not satisfy classical statements in the knowledge base are removed from the set of possible valuations $\mathcal{U}$. Then the power set of the remaining valuations is computed, excluding the empty set. Let's call this $\mathfrak{G}$. Each subset s of $\mathfrak{G}$ is considered and, starting with the arrangement where every valuation is taken as most typical, each set of equivalent ranked interpretations is considered for that set of valuations. This is done by moving one valuation up a level at every step. Ranked interpretations are equivalent in this case if they have the same structure and are incomparable to each other with respect to $\trianglelefteq_{PT}$. Starting with the ranked interpretation where every possible valuation is on L_0 is motivated by the fact that this would be the minimal ranked interpretation under $\trianglelefteq_{\text{PT}}$. A ranked interpretation $\mathscr{R}_1$ is considered better than another, $\mathscr{R}_2$, if $\mathscr{R}_1 \trianglelefteq_{\text{PT}} \mathscr{R}_2$.

An arrangement can be represented as a number where each digit represents a valuation, and its value the level it is on. Suppose we have three interpretations v_1, v_2 and v_3. We would start with the arrangement 000. Then try 001, 010 and 100, and so on and so forth until the arrangements we are trying are a linear ordering of interpretations and there are no more ways to move valuations up. As a rule there can not be an empty level between two levels, since this is just equivalent to removing that level entirely. To denote this let A_i be the set of arrangements in the ith iteration.

Every ranked interpretation is checked for satisfaction of the typicality knowledge base. If it does indeed satisfy the knowledge base, it is a model for it. This check is done by computing whether for every valuation in the interpretation, it satisfies every sentence in the knowledge base classically and with respect to the typicality operator. When an atom has the typicality operator attached to it, e.g. let's say in the sentence $\bullet a \rightarrow b$, and we are checking a valuation v. We also check whether, in the valuation we are checking, it is indeed the typical case. This simply involves checking what the lowest level an interpretation where a is true exists. If it is the typical, i.e. lowest level for $\bullet a$, we can evaluate it as a classical statement, if not then $\bullet a$ is false and we evaluate by replacing it with $\bot$.

The set of minimal models for each subset s is denoted $\mathfrak{R}_s$. When a ranked model is found: only the equivalent arrangements are considered and the search stops for that set of valuations. The justification for this is that any arrangements in a subsequent iteration will be worse than the one already found. Once a model is found for a set of valuations, that set is removed from $\mathfrak{G}$. When a model for a subset of size n is found, it is compared to every previously found model. A model with $n-1$ valuations in it can be worse than one with n valuations in it, or it can be incomparable. However, it cannot be better than it since at least 1 valuation will be on level

infinity. For any model found, let's call it $\mathscr{R}$, if it worse than a previous model $\mathscr{R}'$, i.e. $\mathscr{R}' \trianglelefteq_{\text{PT}} \mathscr{R}$, then it is also removed from the set of minimal models.

Once every possible subset has been checked the algorithm terminates and returns $\Re_{\mathcal{K}}$, a set of minimal models for a given knowledge base $\mathcal{K}$.

This algorithm is not very efficient for finding models, as it requires going through every possible subset of a set of valuations. This means it is checking exponentially many sets for models. Similarly when looking for minimal arrangements, the number of arrangements will blow up when the number of valuations being arranged grows.

Once the minimal models have been found, to check entailment for a statement is to check whether in every valuation in every ranked model, the statement is entailed.

For a detailed version of the algorithm used to find minimal models, see Appendix A.

6 IMPLEMENTATION

The main focus of this project is the implementation of PT Entailment in code. This essentially comes down to building a wrapper for a SAT-Solver, since the SAT-solver can check the satisfiability of a series of statements. To do this my partner and I chose to use the same SAT-solver: MiniSAT [10] and to code in Python. This SAT-solver is a suitable fit for our purposes, and to interface with it we used a pre-existing Python library called PyMiniSolvers, licensed under the MIT license[1]. The code for the implementation can be found on GitHub. [2]The main components of this implementation are detailed as follows.

6.1 Language Representation

The input format for formulas was chosen to a readable approximation of the sentences of PTL. The following table shows the representation of each symbol:

Symbol	*Representation*
$\wedge$	&
$\vee$	|
$\neg$	-
$\rightarrow$	>
$\bullet$	*

An example of such a sentence would be `*p > -f`. These sentences are input as strings. They should be properly bracketed to read unambiguously, otherwise they are processed as being bracketed associatively from right to left. The user should input their knowledge base as a series of strings each representing a sentence, as well as the statement that is being checked for entailment.

Initially in the implementation, in order to represent typicality of a sentence $\bullet\alpha$ instead of just an atom $\bullet p$, the sentence was instead replaced by a new atom q and the new information that says $\alpha \leftrightarrow q$ added to the knowledge base. This approach removes the possibility of nested typicality operators. For example, if we have `*(p & *q)`

[1]https://github.com/liffiton/PyMiniSolvers
[2]https://github.com/AndrewHoweEly/PTR-PT-Entail

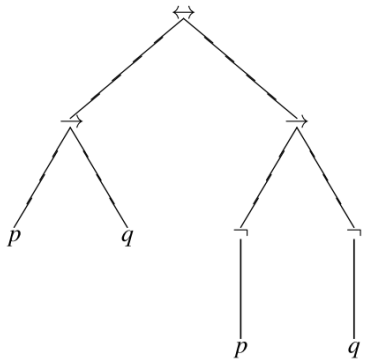

Figure 2: A Formula Tree from Ben-Ari [1]

we would introduce a new variable `x` and change the sentence to be `*x` and add the new sentences `x > (p & *q)` and `(p & *q) > x`. Unfortunately, this naive solution had the side effect of adding more variables to the situation and makes entailing sentences with typicality of more than just a single atom impossible.

The other data structures used were 2D arrays to represent a Ranked Model as well as sets and lists to represent sets of valuations. A knowledge base is simply a list of sentences. Every valuation is represented as a binary string, with a list of variables kept also used to keep the ordering of the valuations in the string. For example with a variable list `[p, q]` the valuation 10 would represent $\{p, \neg q\}$.

6.2 SAT-Solving

Calls to the SAT solver in the MiniSolvers library must be in CNF as in Section 2.3, with each clause being added separately to the solver. A new variable needs to be created in the solver class for each variable used in the clauses. An example set of clauses would be: [1, 2], [-2]. Logically this is equivalent to $(x_1 \vee x_2) \wedge \neg x_2$.

This format only allows for sentences containing *and*s and *or*s. Implementing this using only string manipulation proved difficult, since some of the conversions require the negation of statements and the rearrangement or propagation of *and*s and *or*s. To solve this the string sentences are read into a binary tree format. This idea was taken from the book on propositional logic by Ben-Ari [1]. The root of each tree is the operator with highest precedence and the left and right nodes are those sentences or atoms on the left or right of each operator. Figure 2 shows an example of such a tree from Ben-Ari.

Several conversions need to be done to each formula tree. The first is to convert an implication $p \rightarrow q$ into $\neg p \vee q$. This is done for each > node in the tree. In order to convert to CNF; instances where the disjunction of a conjunction takes place - an *or* of *and*s - must be detected and converted so that the & takes precedence. This is done recursively down the tree with the base case being the same as Equation 1. Other tree operations include propagating negations across conjunctions and disjunctions using De Morgan's laws, e.g. $\neg(p \wedge q) \leftrightarrow \neg p \vee \neg q$. The order of the conversion steps is: first convert implications, then propagate negations and lastly convert *or*s of *and*s. Once the tree has been processed it can be converted

into SAT form simply by producing an inorder string of nodes in the tree, since it will now be in the correct CNF format. The sentence is split on & and each substring becomes its own clause. The list of variables is used to align the variables used in the SAT Solver with the relevant atoms.

6.3 Checking Satisfaction

An important step in the algorithm for finding minimal models is to check whether an interpretation satisfies a knowledge base $\mathcal{K}$. For a single valuation this means checking whether the conjunction of the valuation, itself a conjunction of literals e.g. 11 or $\{p, q\}$ becomes $p \wedge q$, and $\mathcal{K}$ is satisfiable. This is easy to pass to the SAT-Solver. To determine whether a ranked interpretation $\mathscr{R}$ satisfies $\mathcal{K}$, we check whether every valuation in $\mathscr{R}$ satisfies the $\mathcal{K}$ with respect to the ranked structure.

For each valuation, a new knowledge base is created using the ranks. For every sentence in $\mathcal{K}$, it is checked for statements of the form $\bullet\alpha$, where α is either a sentence, a single atom or the sentence which is just the negation of one atom. If the valuation is on the same level as the most typical occurrence of α, i.e. the lowest level with a valuation where α is true, we simply remove the occurrence of the typicality operator and check sentence classically. If it is not the most typical level, we replace $\bullet\alpha$ with $p \wedge \neg p$, where p is an atom in the variable list, since this is equivalent to false. Where α is just equivalent to p or $\neg p$ for some atom p, we only have to see where the lowest level a valuation with 1 or 0 respectively in p's position is. Otherwise, it is a task for the SAT-Solver method to check each valuation for satisfaction of α.

What is happening here is that on the most typical level we can simply check whether a sentence is satisfied classically, without regard to the typicality. But when the typicality is false we cannot evaluate the sentence as it was classically. Suppose we have the statement $\bullet b \rightarrow f$, *typical birds fly*, and an interpretation that does not correspond to $\bullet b$. Then, replacing the left hand side with false, the statement as a whole would be true, since we are not looking at the most typical birds.

If every valuation satisfies its new knowledge base, the entire ranked interpretation is indeed a ranked model.

6.4 Generating PT Minimal Models

The algorithm to find PT minimal models starts by generating the power set of the set of all valuations. Then valuations which do not satisfy the classical statements in the knowledge base, i.e. those without $\star$, are thrown out. The largest subset is considered first and for each subsequent subset the same procedure follows. Possible arrangements of valuations in a ranked model are generated recursively and are tested to see whether they form a model.

To generate possible arrangements, each is represented as a string of digits where every digit represents the level of a valuation. An algorithm to produce these one iteration at a time was used. Each subsequent iteration returns all the ways of adding 1 to exactly one digit in every arrangement in the previous iteration. The procedure starts with the arrangement 0..0 and iterates until no new arrangements are created, discounting repeated arrangements as well invalid arrangements, i.e. those with empty levels.

Each set of valuations is converted into a ranked interpretation corresponding to an arrangement in the current iteration of arrangements. Then, if that ranked interpretation is a model for the knowledge base it is added to the list of models found. Only those which are PT-minimal are kept and returned. This is done by comparing each model found to those previously found using the $\trianglelefteq_{\text{PT}}$ relation. A ranked model is preferred if every possible valuation is on an equal or lower level in the model to those in another model. Those valuations not in a ranked interpretation are effectively on level ∞.

A possible optimisation that was considered was to discard every subset of a set of valuations if a ranked model is found for that subset. However, this resulted in not every minimal model being returned, so was not viable. See Section 6.6 for an example that shows a minimal model formed from a subset of another model. This initial algorithm looked at the maximum set of subsets before throwing each set out. However, since we do not discard any subsets we are in effect just looking at every set and are only going in decreasing order of size.

6.5 Computing Entailment

Entailment is implemented by repeatedly checking classical entailment for each valuation in a model as well as the additional typicality check. The way classical entailment is checked is by adding the negation of the sentence being checked to the rest of the knowledge base and entered into the SAT-solver. If this returns false, it means the sentence is indeed entailed. This is because $\mathcal{K} \models \alpha$ is equivalent to $\mathcal{K} \wedge \neg\alpha$ being unsatisfiable, since there is no model that satisfies both the knowledge base and the negation of the sentence.

To check if a knowledge base PT-entails a sentence, first find all PT-minimal models. Then for every minimal model, every valuation on every level is checked to see if it entails that sentence with respect to the ranked model. The sentence is modified to remove typicality instances following the same scheme as in Section 6.3, which comes down to checking if an typicality sentence $\bullet\alpha$ is typical on each valuation's level and if that valuation entails α. If there is a valuation on a level that does not entail its modified sentence then the entire check returns false.

6.6 Testing

Testing was done on small knowledge bases. These are just toy examples, but the results of the entailment algorithm match the theoretical expectation. For example, with a knowledge base of $\{\bullet p \rightarrow \neg f, \bullet b \rightarrow f, p \rightarrow b\}$ we generate the ranked model found in Figure 1. A property of PT entailment is that if we only consider statements of the form $a \mid\sim b$ as in that example, we get the same single ranked model as in LM Entailment. From this we can successfully entail $\bullet b \rightarrow \neg p$.

Results follow for small knowledge bases with a small number of atoms, usually producing only one minimal model. Testing against the results from my partner, Guy Green, gives the same answers for these small problems. This shows it is correct for small knowledge bases.

However, due to the nature of checking each possible ranked interpretation, the complexity of the algorithm is extremely large. Consider the fact that for each sentence with n atoms there are 2^n

interpretations. Then the number of possible non-empty subsets of these interpretations is $2^{2^n} - 1$. And each of these subsets must be checked for all possible arrangements, which again can be exponentially many. This makes the algorithm very inefficient for large knowledge bases.

An example from the Booth 2015 paper is $\mathcal{K} := \{\bullet\top \rightarrow (\neg p \wedge \neg r), \bullet p \rightarrow \bullet\neg f, \bullet r \rightarrow \bullet f\}$, in other words "the most typical things are neither penguins nor robins, typical penguins are typical non-flying birds, and typical robins are typical flying birds". This example can be modeled, converting the typicality of $\top$ into $p \vee \neg p$, in the implementation as:

```
["(*(p|-p))>(-p&-r)", "*p>*-f", "*r>*f"].
```

The resulting minimal models generated match with those PT minimal models found in the paper:

$\mathscr{R}_1$

L_0 :	$\{\neg f, \neg p, \neg r\}$	$\{f, \neg p, \neg r\}$	

$\mathscr{R}_2$

L_2 :		$\{f, p, \neg r\}$	
L_1 :	$\{\neg f, \neg p, \neg r\}$		$\{\neg f, p, \neg r\}$
L_0 :		$\{f, \neg p, \neg r\}$	

$\mathscr{R}_3$

L_2 :		$\{\neg f, \neg p, r\}$	
L_1 :	$\{f, \neg p, r\}$		$\{f, \neg p, \neg r\}$
L_0 :		$\{\neg f, \neg p, \neg r\}$	

Figure 3: PT Minimal Models

Note that $\mathscr{R}_1$ is the model produced by LM Entailment and its set of valuations is a subset of $\mathscr{R}_3$.

Some expected results of entailment from Booth et al. [2] are that $\bullet\neg p \rightarrow \neg r$ is entailed by the knowledge base, or more specifically from the PT Minimal models. This entailment is true in the implementation so we can reason that *typical non-penguins are not robins*. Two more examples are that neither $\bullet\neg p \rightarrow \neg f$ nor $\bullet(\neg p \wedge f) \rightarrow \neg r$ are entailed by the knowledge base. The implementation correctly produces false for both of these and they are therefore not in $Cn_{\text{PT}}(\mathcal{K})$.

These tests produced the correct minimal models and the correct entailment results. This means that the implementation is indeed a successful proof of concept for PT Entailment, even though it is not a highly efficient implementation.

7 CONCLUSIONS

The focus of this report was the implementation of PT Entailment for the logic PTL. The approach was to first detail what PT Entailment means theoretically, with the necessary background in propositional and defeasible logic provided. Several potential properties that entailment in PTL should satisfy are given, with the result that it is not possible for all of these postulates to be satisfied at once. Instead, PT Entailment satisfies the subset P1-4, P7 and P8-10 of these properties. PT Entailment therefore produces different results to that of LM Entailment. This was indeed shown by the respective implementations.

An implementation of PT Entailment was created that matched the high level theoretical description. This was achieved by creating a wrapper in Python around the MiniSAT SAT Solver and using an open source library to interface with it. This implementation serves as a successful proof of concept for PT Entailment computationally. Testing on small knowledge bases gives the same results as can be derived theoretically. Specifically, using examples from the literature produced the same minimal models and entailment consequences. However, the implementation also proved inefficient as the complexity of the algorithm is exponential. This stems not only from the specific implementation but also its theoretical description.

Future work that could be done in this area would be to optimise the algorithm for finding PT minimal models on a theoretical as well as computational level.

8 ACKNOWLEDGEMENTS

Thank you to my partner, Guy Green, and my supervisor, Professor Tommie Meyer, for their help.

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, 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.
[4] 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.
[5] 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.
[6] Weiwei Gong and Xu Zhou. 2017. A survey of SAT solver. *AIP Conference Proceedings* 1836, 1 (2017), 020059. https://doi.org/10.1063/1.4981999 arXiv:https://aip.scitation.org/doi/pdf/10.1063/1.4981999
[7] Sarit Kraus, Daniel Lehmann, and Menachem Magidor. 1990. Nonmonotonic reasoning, preferential models and cumulative logics. *Artificial Intelligence* 44 (1990), 167–207.
[8] Daniel Lehmann and Menachem Magidor. 1992. What does a conditional knowledge base entail? *Artificial Intelligence* 55 (1992), 1–60.
[9] Bart Selman, Henry A Kautz, Bram Cohen, et al. 1993. Local search strategies for satisfiability testing. *Cliques, coloring, and satisfiability* 26 (1993), 521–532.
[10] Niklas Sorensson and Niklas Een. 2005. Minisat v1. 13-a sat solver with conflict-clause minimization. *SAT* 2005, 53 (2005), 1–2.

A PT-MINIMAL ALGORITHM

Procedure PT-minimal($\mathcal{K}$)

Input: $\mathcal{K}$, $\mathcal{U}$
Output: $\mathfrak{R}_{\mathcal{K}}$ //$\mathfrak{R}_{\mathcal{K}}$ is the set of the PT-minimal models of $\mathcal{K}$

$\mathcal{U} := \{v \in \mathcal{U} \mid v \Vdash \alpha, \alpha \in \mathcal{K} \cap \mathcal{L}\}$ // $\mathcal{U}$ is restricted to valuations that satisfy the classical statements in $\mathcal{K}$;
$\mathfrak{R}_{\mathcal{K}} := \emptyset$;
$\mathfrak{G} := \mathscr{P}(\mathcal{U}) \setminus \emptyset$;
repeat
  foreach $s \in \max_{\subseteq}(\mathfrak{G})$ **do**
    $\mathfrak{R}_s := \emptyset$;
    $i = 0$;
    $A_0 := \{$"0…0"$\}$ // A is the set of arrangements to be considered;
    repeat
      foreach $a \in A_i$ **do**
        if $\mathscr{R}_a \Vdash \mathcal{K}$ **then**
          $\mathfrak{R}_s := \mathfrak{R}_s \cup \mathscr{R}_a$;
      $i := i + 1$;
      if $A_i = \emptyset$ **then**
        break
    until $\mathfrak{R}_s \neq \emptyset$;
    foreach $\mathscr{R} \in \mathfrak{R}_{\mathcal{K}}$ **do**
      foreach $\mathscr{R}' \in \mathfrak{R}_s$ **do**
        if $\mathscr{R} \trianglelefteq_{\mathrm{PT}} \mathscr{R}'$ **then**
          $\mathfrak{R}_s := \mathfrak{R}_s \setminus \mathscr{R}'$;
    $\mathfrak{R}_{\mathcal{K}} := \mathfrak{R}_{\mathcal{K}} \cup \mathfrak{R}_s$;
    $\mathfrak{G} := \mathfrak{G} \setminus s$;
until $\mathfrak{G} = \emptyset$;
return $\mathfrak{R}_{\mathcal{K}}$