A Defeasible Reasoning Approach for Description Logic Ontologies

Kody Moodley
Centre for Artificial Intelligence Research
kmoodley@csir.co.za

Thomas Meyer
Centre for Artificial Intelligence Research
tmeyer@csir.co.za

Ivan José Varzinczak
Centre for Artificial Intelligence Research
ivarzinczak@csir.co.za

ABSTRACT

Classical reasoning for logic-based KR (Knowledge Representation) systems is in general, *monotonic*. That is, there is an assumption in these systems that there is complete information about a domain. This means that they generally cannot deal with any new information arising which contradicts with the current information. This is not an appropriate model for reasoning in many applications. Therefore, alternative *non-monotonic* systems have been investigated which can reason under uncertainty or with incomplete information. *Defeasible* reasoning is one particular model for implementing non-monotonic reasoning. It is concerned with representing and reasoning with defeasible (non-strict) facts about a domain. The defeasible counterpart of the strict fact: "All birds fly" is the defeasible fact: "*Most* birds fly" (or the alternative phrasing "Birds *usually* fly"). We discuss two approaches for defeasible reasoning in the family of logic-based KR languages known as Description Logics (DLs). They are applicable to particular extensions of DLs that allow for the statement of defeasible sentences similar to the aforementioned examples. The approaches are known as *prototypical* reasoning and *presumptive* reasoning and are both rooted in the notion of *Rational Closure* developed by Lehmann and Magidor for an extension of propositional logic. Here we recast their definitions in a DL context and define algorithms for prototypical and presumptive reasoning for DL knowledge bases (also called DL ontologies) that may contain defeasible sentences. In particular, we present a plug-in for the Protégé ontology editor which implements these algorithms for OWL ontologies - the Web Ontology Language (OWL) is a formal standard of languages whose semantic basis is identical to that of DLs. Our plug-in, *RaMP*, allows the modeller to indicate defeasible information in OWL ontologies and perform logical inferencing to determine what defeasible conclusions one can draw from these ontologies.

Categories and Subject Descriptors

I.2.4 [**Computing Methodologies**]: Artificial Intelligence—*KR Formalisms and Methods*

General Terms

Algorithms, Design, Languages, Theory

SAICSIT '12, October 01-03 2012, Pretoria, South Africa

Keywords

Defeasible reasoning, Rational closure, OWL, Protégé

1. INTRODUCTION

An artefact which formally describes a conceptual understanding of a particular domain of interest is known as an *ontology*. There are many uses for ontologies in computer information systems and also numerous formalisms that are used to capture them. An obvious usage for them is as a tool for visualising (and therefore understanding) how systems and processes work and how their components function together. More recently, with the advent of the *Semantic Web* [4], formal ontologies (those that can be represented by a set of logical statements about the domain) started to look like an attractive means for developing a semantically enriched World Wide Web. The Web Ontology Language (OWL) [20] was developed and the latest specification OWL 2 became a World Wide Web Consortium (W3C) recommended standard in 2009 for representing Semantic Web ontologies. The OWL family of languages are precise, logic-based formalisms. Because of their precise syntax, ontologies expressed using these languages are machine-readable. Moreover, because of the logical basis of the language, it becomes possible to build automated reasoning systems to extract implicit information (knowledge) from these ontologies which is not explicitly stated. A typical reasoning task in such systems is logical inference. For example, if we are given the two statements: "all birds fly" and "tweety is a bird" then a reasoner could infer that "tweety flies".

Description Logics (DLs) [2] are another widely accepted and appropriate class of knowledge representation languages to represent and reason about ontologies. Although they are distinct from the OWL family of languages (also called fragments), from a purely logical perspective they actually form the basis of most OWL fragments. Description Logics as a family of ontology languages provide a good balance between expressive power (the type of knowledge you can represent in the language) and computational complexity of reasoning [5]. Since these languages have a precise syntax and semantics (like OWL), they allow one to represent an ontology purely as a set of logical statements about the domain. Such a view of an ontology is called a knowledge base (KB). Again it is possible to build inference engines (DL reasoners [24, 25, 26]) for inferring implicit information from these KBs. Furthermore, such reasoners for performing logical inference have grown increasingly powerful and sophisticated in the last decade.

In the setting we have established, i.e, in standard DL-based reasoning systems, the notion of *entailment* (logical inferencing) is *monotonic*. Monotonicity is a property of KR systems that are built upon classical logics. It specifies that knowledge is always 'incremental'. That is, adding to (or logically strengthening) the information in a KB cannot result in any previously known conclusions being retracted from the KB. In classical logics, this one

intuitive notion of monotonic behaviour is exhibited on two levels. Firstly, on the meta-level where if a statement φ follows logically from a KB $\mathcal{K}$ then φ also follows from any superset of $\mathcal{K}$ and, secondly, on the object level from $\alpha \rightarrow \beta$ it follows that $\alpha \wedge \gamma \rightarrow \beta$ for any γ. We are particularly interested in this last context which displays the monotonicity. If we let α represent the statement "I am a bird", β "I can fly" and γ "I am a penguin" then the last notion of monotonicity becomes problematic for our example. This is because the property essentially requires that the statement: "If I am a bird then I can fly" also implies the statement: "If I am a bird and I am a penguin then I can fly". It is clear that this conclusion is not reasonable in all domains. For example, in a biological domain there is no evidence currently to suggest that penguins can fly. Therefore in this setting, even though penguins are classified as birds, we do not necessarily want our representation of a penguin to inherit *all* the properties of a bird. Especially the property of being able to fly.

Thus it turns out that there are applications in which monotonicity is undesirable, i.e., *non-monotonic* reasoning is required. A typical scenario is when one needs to model *exceptions* in a domain. As we have demonstrated in the penguin example, in a particular domain of interest it might be desirable to express logical statements that you want to hold in *typical* situations. That is, we may want to say that birds *typically* fly and there are cases in which birds do *not* fly (for example when we have a bird which also happens to be a penguin). These non-typical cases represent exceptions and are more rare but nevertheless entirely possible scenarios. The statement that birds typically or usually fly is an example of *defeasible* information. The broad approach to reasoning with KBs that contain defeasible information is known as *defeasible reasoning* and is a popular way to introduce non-monotonic reasoning behaviour into knowledge representation and reasoning systems. We argue that it is very useful in numerous domains to develop a defeasible reasoning approach to capture intuitions as in the above examples. It would also be useful to have a robust system in the DL setting that uses a purely logical approach but is also capable of catering for exceptions as demonstrated above. Lehmann and Magidor [19] provided such an approach in the propositional logic context and Britz et al. [8] provided an extension thereof for DLs.

The main goals of this paper are: (*i*) to give a flavour or intuition for an approach we have developed to do defeasible reasoning in the DL context, (*ii*) to present two algorithms for implementing this approach and (*iii*) to demonstrate a preliminary version of a Protégé [16] plug-in which demonstrates our algorithms and approach for defeasible reasoning with OWL ontologies. The plug-in is called *RaMP* and is discussed in more detail in Section 5. Most of the theoretical notions presented in this paper are extensions of the techniques adopted by Lehmann and Magidor [19] who developed their approach for an extension of propositional logic. We use the semantic foundation by Britz et al. [8] as a basis for our approach.

Throughout this paper we shall adopt the convention of presenting our theoretical research with respect to DLs with the understanding that the findings are easily translatable to a corresponding OWL fragment. Similarly when it comes to describing our practical implementation we do this in the context of OWL because the tool we have built is for use with OWL ontologies. The main reason for this is because of the good infrastructure and support for OWL-related tools in the community.

2. PRELIMINARIES

In this section we give a short introduction to Description Logics as well as defeasible reasoning approaches which are relevant to ours. We begin by giving the syntax and semantics of the popular DL known as $\mathcal{ALC}$. We focus on $\mathcal{ALC}$ because it has a good trade-off between expressivity and computational complexity and is therefore appropriate for most application domains.

2.1 The Description Logic $\mathcal{ALC}$

DLs are decidable fragments of first-order logic with interesting properties and a variety of applications, notably the formalisation of ontologies. There is a whole family of description logics, an example of which is $\mathcal{ALC}$ and on which we shall focus in the present paper.[1]

$\mathcal{ALC}$ concept syntax:

The language of $\mathcal{ALC}$ is built upon a finite set of atomic *concept names* $\mathsf{N}_{\mathscr{C}}$ and a finite set of *role names* $\mathsf{N}_{\mathscr{R}}$ such that $\mathsf{N}_{\mathscr{C}} \cap \mathsf{N}_{\mathscr{R}} = \emptyset$. An atomic concept is denoted by A, possibly with subscripts, and a role name by r, possibly with subscripts. $\top$ denotes the *top concept* and $\bot$ the *bottom concept*. Complex concepts are denoted by $C, D, \ldots$, possibly with subscripts, and are built using the constructors $\sqcap$ (concept conjunction), $\sqcup$ (concept disjunction), $\neg$ (complement), $\exists$ (existential restriction) and $\forall$ (universal restriction) according to the rule

$$C ::= A \mid \top \mid \bot \mid \neg C \mid C \sqcap C \mid C \sqcup C \mid \exists r.C \mid \forall r.C$$

We let $\mathcal{L}$ denote the set of all $\mathcal{ALC}$ concepts. If we are in a medical domain we may use $\mathcal{ALC}$ concept names to capture terms that are relevant in this context. For example, if we are interested in viral diseases and bacterial infections we can use concept names such as $\mathsf{Meningitis}$, $\mathsf{BacterialMeningitis}$, $\mathsf{ViralDisease}$ and $\mathsf{FatalInfection}$.

$\mathcal{ALC}$ concept semantics:

The semantics of $\mathcal{ALC}$ is the standard set theoretic Tarskian semantics. An *interpretation* is a structure $\mathcal{I} = \langle \Delta^{\mathcal{I}}, \cdot^{\mathcal{I}} \rangle$, where $\Delta^{\mathcal{I}}$ is a non-empty set called the *domain*, and $\cdot^{\mathcal{I}}$ is an *interpretation function* mapping concept names A to subsets $A^{\mathcal{I}}$ of $\Delta^{\mathcal{I}}$, and mapping role names r to binary relations $r^{\mathcal{I}}$ over $\Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$:

$$A^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}},\ r^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$$

Given an interpretation $\mathcal{I} = \langle \Delta^{\mathcal{I}}, \cdot^{\mathcal{I}} \rangle$, $\cdot^{\mathcal{I}}$ is extended to interpret complex concepts in the following way:

$$\begin{gathered}
\top^{\mathcal{I}} = \Delta^{\mathcal{I}}, \\
\bot^{\mathcal{I}} = \emptyset, \\
(\neg C)^{\mathcal{I}} = \Delta^{\mathcal{I}} \setminus C^{\mathcal{I}}, \\
(C \sqcap D)^{\mathcal{I}} = C^{\mathcal{I}} \cap D^{\mathcal{I}}, \\
(C \sqcup D)^{\mathcal{I}} = C^{\mathcal{I}} \cup D^{\mathcal{I}}, \\
(\exists r.C)^{\mathcal{I}} = \{x \in \Delta^{\mathcal{I}} \mid \text{for some } y, (x,y) \in r^{\mathcal{I}} \text{ and } y \in C^{\mathcal{I}}\}, \\
(\forall r.C)^{\mathcal{I}} = \{x \in \Delta^{\mathcal{I}} \mid \text{for every } y, (x,y) \in r^{\mathcal{I}} \implies y \in C^{\mathcal{I}}\}
\end{gathered}$$

For example, $\mathsf{Meningitis}^{\mathcal{I}}$ can denote the set of all instances of the meningitis infection in the domain. Intuitively, this may represent all the strains of meningitis that we know of. Similarly, $(\mathsf{Meningitis} \sqcap \mathsf{ViralDisease})^{\mathcal{I}}$ denotes the set of all objects in our domain which belong to *both* $\mathsf{Meningitis}^{\mathcal{I}}$ and $\mathsf{ViralDisease}^{\mathcal{I}}$. Intuitively this may represent all those strains of meningitis that are also viral diseases. We can give the intuitive meanings for the other $\mathcal{ALC}$ concepts in a similar way.

$\mathcal{ALC}$ axiom syntax:

Given $C, D \in \mathcal{L}$, $C \sqsubseteq D$ is a *subsumption statement*, and it is read "C is subsumed by D". $C \equiv D$ (called an *equivalence statement*) is an abbreviation for both $C \sqsubseteq D$ and $D \sqsubseteq C$. An *$\mathcal{ALC}$ TBox* $\mathcal{T}$ is a finite set of subsumption statements.

[1]For the reader not conversant with Description Logics but familiar with modal logics, there are results in the literature relating some families of description logics to systems of modal logic. For example, a well-known result is the one linking the DL $\mathcal{ALC}$ with the normal modal logic K [23].

An example of an $\mathcal{ALC}$ subsumption is BacterialMeningitis $\sqsubseteq$ Meningitis.

$\mathcal{ALC}$ axiom semantics:

An interpretation $\mathcal{I}$ *satisfies* $C \sqsubseteq D$ (denoted $\mathcal{I} \Vdash C \sqsubseteq D$) if and only if $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$. $\mathcal{I} \Vdash C \equiv D$ if and only if $C^{\mathcal{I}} = D^{\mathcal{I}}$. $\mathcal{I}$ *satisfies* BacterialMeningitis $\sqsubseteq$ Meningitis if and only if every object in our domain that belongs to BacterialMeningitis$^{\mathcal{I}}$ also belongs to Meningitis$^{\mathcal{I}}$. Intuitively this means that every strain of bacterial meningitis is also a strain of meningitis.

Entailment:

$C \sqsubseteq D$ is (classically) *entailed* by a TBox $\mathcal{T}$, denoted $\mathcal{T} \models C \sqsubseteq D$, if and only if every interpretation $\mathcal{I}$ which satisfies all elements of $\mathcal{T}$, also satisfies $C \sqsubseteq D$. If $\mathcal{T}$={BacterialMeningitis $\sqsubseteq$ Meningitis, Meningitis $\sqsubseteq \neg$FatalInfection}, a consequence of $\mathcal{T}$ is $\mathcal{T} \models$ BacterialMeningitis $\sqsubseteq \neg$FatalInfection. Intuitively, if every strain of bacterial meningitis is a strain of meningitis and if every strain of meningitis is not fatal then every strain of bacterial meningitis is also not fatal.

For more details on Description Logics in general and on $\mathcal{ALC}$ in particular, the reader is encouraged to read the Description Logic handbook [2].

2.2 Defeasible Reasoning

In this section we give a brief overview of the defeasible reasoning approach developed by Kraus, Lehmann and Magidor (often called the KLM approach) [17, 19] for their extension of propositional logic.

There are various approaches for introducing non-monotonic reasoning capabilities in logic-based knowledge representation systems. Among these, the KLM approach has been particularly successful due to its elegance and robustness and many extensions [22, 13, 6, 9, 7, 8] of this work have been proposed in the literature recently. The KLM approach essentially enriches propositional logic with a defeasible 'implication' operator ($\mid\!\sim$). This operator allows one to write down *defeasible implication* statements (also called *conditional assertions*) of the form $\alpha \mid\!\sim \beta$, where α and β are propositional formulas. The sentence $\alpha \mid\!\sim \beta$ intuitively means that in those situations where α is *typically* true, β is also true (for the precise semantics the reader should consult the provided references).

Given a defeasible logic, such as the KLM-extension of propositional logic or the aforementioned extensions thereof, and a *conditional KB* (a set of conditional assertions) it is very important for reasoning purposes to define precisely what kind of inferences you can make from a conditional KB. Lehmann et al. characterise the notion of *rational closure* [19, Section 5] and motivate this notion to be a suitable description for what you can conclude from a conditional KB. The name given to the reasoning approach that computes rational closure for conditional KBs is *prototypical reasoning*. Prototypical reasoning is seen as a conservative answer to the question of what you can conclude from a conditional KB.

After defining prototypical reasoning Lehmann and colleagues also began to explore other notions of non-monotonic reasoning as well as extensions to prototypical reasoning. One such (venturous) extension is known as *presumptive reasoning* [18]. Any inference that follows from a conditional KB using prototypical reasoning will necessarily also follow using presumptive reasoning while the converse is not necessarily true.

3. DEFEASIBLE REASONING FOR DLS

The first part of this section gives an overview of the defeasible subsumption operator [8] and the issue of what entailment means in the context of defeasible KBs. The remainder of the section is concerned with our contributions of algorithms (based on the semantic adaptations that were required to move to DLs) for computing or performing defeasible reasoning for DLs.

In the work by Britz et al [8], the authors propose a defeasible extension of $\mathcal{ALC}$. The defeasibility in this logic is introduced through a *defeasible subsumption* operator ($\sqsubseteq\!\!\!\sim$). This operator is 'supraclassical' to the classical subsumption operator ($\sqsubseteq$) in the sense that any pair of DL concepts that are related via $\sqsubseteq$, are also necessarily related via $\sqsubseteq\!\!\!\sim$. Intuitively, the semantics for $\sqsubseteq\!\!\!\sim$ states that given a defeasible subsumption axiom $C \sqsubseteq\!\!\!\sim D$ (where C and D may be complex $\mathcal{ALC}$ concepts), then this statement means that the most *typical* C's are also D's (as opposed to *all* C's being D's in the classical case). Only a semantics for defeasible *subsumption* is explicitly provided by the approach (although defeasible *equivalence* follows trivially as well) and the approach is currently applied to *$\mathcal{ALC}$ TBoxes* but we intend to extend the approach for *$\mathcal{ALC}$ ABoxes* as well.

Note that we do not state here the *precise* semantics of the $\sqsubseteq\!\!\!\sim$ operator because we are not concerned with its suitability for capturing a good notion of defeasibility. For an explanation of why this is in fact the case we refer the reader to the work by Lehmann and Magidor [19]. They argue this for their propositional version of this operator. For a precise definition of the semantics for $\sqsubseteq\!\!\!\sim$ we refer the interested reader to the work by Britz et al. [8]. We take a more proof theoretic approach in this paper to describe how we can build a reasoning system which behaves according to the defined semantics by Britz et al.

We can nevertheless state the integrity of $\sqsubseteq\!\!\!\sim$ and its logical underpinning by showing that its behaviour is determined by a series of logical properties [19] which we adapt here for DLs.

Given $\mathcal{ALC}$ concepts C, D, $C \sqsubseteq\!\!\!\sim D$ is a *defeasible subsumption statement*, and it is read "C is subsumed by D usually follows". $\sqsubseteq\!\!\!\sim$ is a binary relation on $\mathcal{L} \times \mathcal{L}$ where $\mathcal{L}$ is the set of all possible $\mathcal{ALC}$ concepts. $\sqsubseteq\!\!\!\sim$ is said to be a *preferential subsumption relation* if and only if it satisfies the following set of properties:

$$\text{(Ref)}\quad C \sqsubseteq\!\!\!\sim C \qquad \text{(LLE)}\quad \frac{\models C \equiv D,\ C \sqsubseteq\!\!\!\sim E}{D \sqsubseteq\!\!\!\sim E}$$

$$\text{(And)}\quad \frac{C \sqsubseteq\!\!\!\sim D,\ C \sqsubseteq\!\!\!\sim E}{C \sqsubseteq\!\!\!\sim D \sqcap E} \qquad \text{(Or)}\quad \frac{C \sqsubseteq\!\!\!\sim E,\ D \sqsubseteq\!\!\!\sim E}{C \sqcup D \sqsubseteq\!\!\!\sim E}$$

$$\text{(RW)}\quad \frac{C \sqsubseteq\!\!\!\sim D,\ \models D \sqsubseteq E}{C \sqsubseteq\!\!\!\sim E} \qquad \text{(CM)}\quad \frac{C \sqsubseteq\!\!\!\sim D,\ C \sqsubseteq\!\!\!\sim E}{C \sqcap D \sqsubseteq\!\!\!\sim E}$$

where Ref is short for Reflexivity, RW for Right Weakening and CM for Cautious Monotonicity. If, in addition to these preferential properties, $\sqsubseteq\!\!\!\sim$ also satisfies the following Rational Monotonicity (RM) property, it is said to be a *rational* subsumption relation:

$$\text{(RM)}\quad \frac{C \sqsubseteq\!\!\!\sim D,\ C \not\sqsubseteq\!\!\!\sim \neg E}{C \sqcap E \sqsubseteq\!\!\!\sim D}$$

We have illustrated the well-founded logical behaviour of the defeasible subsumption operator $\sqsubseteq\!\!\!\sim$ and now we want to define a notion of entailment for $\sqsubseteq\!\!\!\sim$ that gives us intuitive inferences or conclusions. That is, given an $\mathcal{ALC}$ KB that contains defeasible subsumption statements like $C \sqsubseteq\!\!\!\sim D$ (a *defeasible KB*), what other statements (axioms) can one conclude from this KB?

Lehmann et al. have motivated why the standard Tarskian-style notion of entailment is not suitable for defeasible KBs like ours. This notion of entailment is characterised as follows: Let $\mathcal{K}$ be a set of defeasible subsumption statements. We take the set of all ordered pairs (C, D) such that $C \sqsubseteq\!\!\!\sim D \in \mathcal{K}$ and call this set $\mathcal{K}_p$, and we let $\sqsubseteq\!\!\!\sim_{all}$ be the collection of sets of *all* $\sqsubseteq\!\!\!\sim$'s that satisfy the seven properties mentioned earlier in this section. If we pick all those $\sqsubseteq\!\!\!\sim_i \in \sqsubseteq\!\!\!\sim_{all}$ such that $\mathcal{K}_p \subseteq \sqsubseteq\!\!\!\sim_i$ then their intersection $\bigcap \sqsubseteq\!\!\!\sim_i$ defines a notion of entailment (called *ranked*

entailment [19]). However, this (Tarskian) definition of entailment is not "enough".

This is so due to two reasons: (*i*) A Tarskian-style entailment relation still defines a consequence relation that is monotonic, falling short of our stated aim of having a consequence relation that can cope with retraction of knowledge previously known, and (*ii*) It is known, both in the propositional case [19, Section 4.2] as well as in the DL one [8, Section 5], that a Tarskian-style entailment relation delivers defeasible inferences that are not, in general, rational, i.e., they do not necessarily satisfy the Rational Monotonicity (RM) property. This motivation for exploring other forms of entailment for defeasible KBs led to the development of the notion of rational closure [19, Section 5]. Rational closure was found to satisfy Rational Monotonicity and argued to be a sensible notion of entailment for defeasible KBs.

Rational closure for $\mathcal{ALC}$ defeasible KBs has been characterised by Britz et al. and the authors present a result [8, Theorem 5] which facilitates the development of an algorithm for computing the rational closure of a defeasible KB. This marks the start of our contribution to the work. We present our implementation of this algorithm (called prototypical reasoning) in the next section.

3.1 Prototypical Reasoning

Prototypical reasoning corresponds exactly to the propositional notion of rational closure, lifted to the DL case. We are not concerned with the semantics here. For the semantics of rational closure in a propositional context, the reader may consult the work of Lehmann and colleagues [19]. For the semantics of rational closure in a defeasible extension of $\mathcal{ALC}$ one can consult the work of Britz et al. [8].

The algorithm for prototypical reasoning, takes as input a subsumption statement φ (also called a *query*) which may be defeasible or classical and a defeasible KB $\mathcal{K}$. The output of the algorithm is **true** if φ follows prototypically from/is in the rational closure of $\mathcal{K}$ (denoted $\mathcal{K} \models_{Prot} \varphi$). Queries and defeasible KBs are currently restricted to subsumption statements for simplicity seeing that (in most cases) classical $\mathcal{ALC}$ TBox axioms can be rewritten as classical $\mathcal{ALC}$ subsumption statements.

The algorithm begins by performing a *classical transformation* of the input KB. Essentially this amounts to rewriting all defeasible statements in the KB as their classical counterparts and all classical statements into a specific normal form. The reason behind this transformation is two-fold: (*i*) It allows classical reasoning techniques to be used on the transformed KB and (*ii*) The normal form of the classical statements differentiates these (from the perspective of the algorithm) from the defeasible statements in the KB. As we shall see later, this last point also makes the next sub-procedure of the algorithm possible (the computation of a ranking of the statements in the KB). The transformation procedure is given below and we give an example to illustrate:

Definition 1. (*transformKB*) Given a defeasible KB $\mathcal{K}$: *transformKB*($\mathcal{K}$) := {*transform*(φ) | $\varphi \in \mathcal{K}$}, where

$$\textit{transform}(\varphi) := \begin{cases} \alpha \sqcap \neg\beta \sqsubseteq \bot, \text{ if } \varphi = \alpha \sqsubseteq \beta, \\ \alpha \sqsubseteq \beta, \text{ if } \varphi = \alpha \mathrel{\tilde{\sqsubseteq}} \beta \end{cases}$$

Example 1. Let $\mathcal{K}$ be the input of transformKB

$$\mathcal{K} = \left\{ \begin{array}{l} \mathsf{Meningitis} \mathrel{\tilde{\sqsubseteq}} \neg\mathsf{FatalInfection}, \\ \mathsf{BacterialMeningitis} \sqsubseteq \mathsf{Meningitis}, \\ \mathsf{ViralMeningitis} \sqsubseteq \mathsf{Meningitis}, \\ \mathsf{BacterialMeningitis} \mathrel{\tilde{\sqsubseteq}} \mathsf{FatalInfection}, \\ \mathsf{Meningitis} \mathrel{\tilde{\sqsubseteq}} \mathsf{ViralDisease} \end{array} \right\}$$

If we execute the procedure in Definition 1 for $\mathcal{K}$ we get $\mathcal{K}^{\sqsubseteq}$:

$$\mathcal{K}^{\sqsubseteq} = \left\{ \begin{array}{l} \mathsf{Meningitis} \sqsubseteq \neg\mathsf{FatalInfection}, \\ \mathsf{BacterialMeningitis} \sqcap \neg\mathsf{Meningitis} \sqsubseteq \bot, \\ \mathsf{ViralMeningitis} \sqcap \neg\mathsf{Meningitis} \sqsubseteq \bot, \\ \mathsf{BacterialMeningitis} \sqsubseteq \mathsf{FatalInfection}, \\ \mathsf{Meningitis} \sqsubseteq \mathsf{ViralDisease} \end{array} \right\}$$

We call $\mathcal{K}^{\sqsubseteq}$ the *classical counterpart* of $\mathcal{K}$.The second procedure of the prototypical algorithm is the computation of a *ranking* of sentences in the KB according to the notion of *exceptionality* [19]. This procedure makes use of a sub-procedure *exceptional* which encodes the notion of exceptionality into the computation of the ranking. Before we specify the pseudocode for the complete procedure we define what we mean by the terms ranking and exceptionality.

Definition 2. (*Ranking*) Let $\mathcal{K}^{\sqsubseteq}$ be the classical counterpart of some defeasible KB: A *ranking* for $\mathcal{K}^{\sqsubseteq}$ is a total preorder on the elements (axioms) in $\mathcal{K}^{\sqsubseteq}$, with axioms higher up in the ordering interpreted as having a higher exceptionality or importance.

Note that a ranking can in practice be implemented as a collection of sets where each element of this collection is a set of sentences (which we call a *rank*) from the KB. Each sentence in a particular set has the same magnitude of importance.

The way we decide which sentences belong to which ranks is calculated according to the notion of exceptionality. Intuitively, a *concept* C is said to be exceptional w.r.t. a defeasible KB $\mathcal{K}$, if it is the case that $\top \sqsubseteq \neg C$ *usually* follows from $\mathcal{K}$. That is, *typically* everything is in $\neg C$, thereby making C an *exception* to this rule. It is important to note that, on the level of sentences in the KB, this notion of exceptionality is different to the notion of exceptionality when we say that a penguin is an "exception" to the rule that all birds fly. A more intuitive term for exceptionality would have been *specificity* because ultimately what we are trying to do is to arrange the sentences in the KB according to how specific or general they are. Nevertheless, we employ the use of the term exceptionality as Lehmann et al. have defined it.

It is also useful to recognise that in the context of our algorithm, checking whether C is exceptional w.r.t $\mathcal{K}$ can be reduced to checking if $\mathcal{K}^{\sqsubseteq}$ *classically* entails $C \sqsubseteq \bot$ where $\mathcal{K}^{\sqsubseteq}$ is the classical transformation of $\mathcal{K}$. Finally, the notion of exceptionality can be extended to *sentences* and *sets* of sentences and we formalise all this in the following definition:

Definition 3. (*Exceptionality*) Let $\mathcal{K}$ be a defeasible KB and let $C, D \in \mathcal{L}$. Also, let $\mathcal{K}^{\sqsubseteq}$ be the classical counterpart of the defeasible KB $\mathcal{K}$:

- C is *exceptional* w.r.t. $\mathcal{K}$ iff $\mathcal{K}^{\sqsubseteq} \models C \sqsubseteq \bot$.
- $C \mathrel{\tilde{\sqsubseteq}} D$ is *exceptional* w.r.t. $\mathcal{K}$ iff C is *exceptional* w.r.t. $\mathcal{K}$.
- $C \sqsubseteq D$ is *exceptional* w.r.t. $\mathcal{K}$ iff $C \sqcap \neg D$ is *exceptional* w.r.t. $\mathcal{K}$.
- $\mathcal{K}' \subseteq \mathcal{K}$, is *exceptional* w.r.t. $\mathcal{K}$ iff every element of $\mathcal{K}'$ and *only* the elements of $\mathcal{K}'$ are exceptional w.r.t. $\mathcal{K}$ (we say that $\mathcal{K}'$ is *more exceptional* than $\mathcal{K}$).

We now specify a sub-procedure called $exceptional(\mathcal{E})$ which computes a more exceptional (specific) subset $\mathcal{E}'$ of some input set of sentences $\mathcal{E}$ according to the last point in Definition 3.

A natural question following from Definition 3 and Procedure $exceptional$ is how this notion is used to decide the full ranking of a defeasible KB. In other words, how do we know if some sentence α is *more* exceptional than another sentence β?

Procedure $exceptional(\mathcal{E})$

Input: Set of sentences, $\mathcal{E}$, from transformed KB
Output: More exceptional subset, $\mathcal{E}'$, of $\mathcal{E}$

$\mathcal{E}' := \emptyset$;
foreach $C \sqsubseteq D \in \mathcal{E}$ **do**
  if $\mathcal{E} \models C \sqsubseteq \bot$ **then**
    $\mathcal{E}' := \mathcal{E}' \cup \{C \sqsubseteq D\}$;
return $\mathcal{E}'$;

To fully appreciate how this procedure influences the ranks of the sentences in the KB one should study how it is used in the complete procedure for computing the ranking of a defeasible KB which we specify below:

Procedure $computeRanking(\mathcal{K})$

Input: Defeasible knowledge base $\mathcal{K}$
Output: The ranking, $\mathcal{D}$, for $\mathcal{K}$

1 $i := 0$; $\mathcal{K}^{\sqsubseteq} :=$ *transformKB*($\mathcal{K}$);
2 $\mathcal{E}_0 := \mathcal{K}^{\sqsubseteq}$; $\mathcal{E}_1 :=$ *exceptional*($\mathcal{E}_0$);
3 **while** $\mathcal{E}_{i+1} \neq \mathcal{E}_i$ **do**
4 $i := i + 1$;
5 $\mathcal{E}_{i+1} :=$ *exceptional*($\mathcal{E}_i$);
6 $n := i$; $max := n + 1$;
7 $\mathcal{D}_{max} := \mathcal{E}_n$; $\mathcal{D} := \{\mathcal{D}_{max}\}$;
8 **for** $j = 1$ *to* n **do**
9 $\mathcal{D}_j := \mathcal{E}_{j-1} \backslash \mathcal{E}_j$;
10 $\mathcal{D} := \mathcal{D} \cup \{\mathcal{D}_j\}$;
11 **return** $\mathcal{D}$;

Recall that the ranking of the KB is computed as a collection of sets of sentences ($\mathcal{D}$ in Procedure $computeRanking$). Each sentence in a particular set from $\mathcal{D}$ shares the same level of exceptionality according to the algorithm. Given the collection of sets $\mathcal{D} = \{\mathcal{D}_1, \ldots, \mathcal{D}_n\}$, $\mathcal{D}_1$ represents the *lowest* rank containing the *least* exceptional (specific) sentences in the KB. Intuitively, because these sentences are seen as the least specific they are the ones which can be "disregarded" with the most confidence when contradicting information is found. On the other end of the spectrum we have $\mathcal{D}_n$ representing the *highest* rank which contains the *most* exceptional (specific) sentences in the KB. Intuitively speaking, these sentences should only be disregarded as a "last resort".

From an algorithmic perspective $\mathcal{D}_n$ is the highest rank but strictly speaking the special rank $\mathcal{D}_{max}$ is the highest. This (possibly empty) rank contains the non-defeasible or classical sentences in the KB. We sometimes refer to these sentences as *hard* in contrast with defeasible statements which we sometimes refer to as *soft*. Hard sentences can *never* be disregarded and should always remain in the KB, hence the special rank for them. The value of max is $n + 1$ which is always one bigger than the number of "normal" ranks. This ensures that the algorithm never "touches" the sentences in this rank because we only wish to manipulate ranks up to $\mathcal{D}_n$.

We give a simple example to illustrate how the ranking of a defeasible KB is computed:

Example 2. Consider the KB $\mathcal{K}$ from Example 1:

$$\mathcal{K} = \left\{ \begin{array}{l} \mathsf{Meningitis} \tilde{\sqsubseteq} \neg\mathsf{FatalInfection}, \\ \mathsf{BacterialMeningitis} \sqsubseteq \mathsf{Meningitis}, \\ \mathsf{ViralMeningitis} \sqsubseteq \mathsf{Meningitis}, \\ \mathsf{BacterialMeningitis} \tilde{\sqsubseteq} \mathsf{FatalInfection}, \\ \mathsf{Meningitis} \tilde{\sqsubseteq} \mathsf{ViralDisease} \end{array} \right\}$$

Line 1 of Algorithm $computeRanking$ computes the classical transformation of $\mathcal{K}$ which is then assigned to $\mathcal{K}^{\sqsubseteq}$. $\mathcal{K}^{\sqsubseteq}$ is shown below:

$$\mathcal{K}^{\sqsubseteq} = \left\{ \begin{array}{l} \mathsf{Meningitis} \sqsubseteq \neg\mathsf{FatalInfection}, \\ \mathsf{BacterialMeningitis} \sqcap \neg\mathsf{Meningitis} \sqsubseteq \bot, \\ \mathsf{ViralMeningitis} \sqcap \neg\mathsf{Meningitis} \sqsubseteq \bot, \\ \mathsf{BacterialMeningitis} \sqsubseteq \mathsf{FatalInfection}, \\ \mathsf{Meningitis} \sqsubseteq \mathsf{ViralDisease} \end{array} \right\}$$

Lines 2 to 5 denote the first phase of the algorithm which incrementally "unpacks" the different subsets of exceptional sentences from $\mathcal{K}^{\sqsubseteq}$. Line 6 captures the indices for the highest and special ranks. Line 7 assigns the special rank as the most exceptional (specific) sentences $\mathcal{E}_n$ which denote the hard sentences. Lines 8-10 "prunes" the sets of unpacked exceptionality sets into a set of ranks (the final ranking). This last step is required because an $\mathcal{E}_i$ is determined with respect to $\mathcal{E}_{i-1}$ and thus $\mathcal{E}_i \subseteq \mathcal{E}_{i-1}$ and so they cannot be ranks.

Returning to our example we recall that $\mathcal{E}_0$ is $\mathcal{K}^{\sqsubseteq}$ (Line 1 of the procedure). On first execution of Lines 2-5 we find that:

$$\mathcal{E}_1 = \left\{ \begin{array}{l} \mathsf{BacterialMeningitis} \sqcap \neg\mathsf{Meningitis} \sqsubseteq \bot, \\ \mathsf{ViralMeningitis} \sqcap \neg\mathsf{Meningitis} \sqsubseteq \bot, \\ \mathsf{BacterialMeningitis} \sqsubseteq \mathsf{FatalInfection} \end{array} \right\}$$

We continue execution of the while loop until we reach a fixed point where $\mathcal{E}_{i+1} = \mathcal{E}_i$. We do not show each computed $\mathcal{E}_i$ for our example but suffice it to say that if the procedure is followed correctly the final ranking for the original KB $\mathcal{K}$ is as follows:

$$\mathcal{D}_{max} = \left\{ \begin{array}{l} \mathsf{BacterialMeningitis} \sqcap \neg\mathsf{Meningitis} \sqsubseteq \bot, \\ \mathsf{ViralMeningitis} \sqcap \neg\mathsf{Meningitis} \sqsubseteq \bot \end{array} \right\}$$

$$\mathcal{D}_2 = \left\{ \begin{array}{l} \mathsf{BacterialMeningitis} \sqsubseteq \mathsf{FatalInfection} \end{array} \right\}$$

$$\mathcal{D}_1 = \left\{ \begin{array}{l} \mathsf{Meningitis} \sqsubseteq \neg\mathsf{FatalInfection}, \\ \mathsf{Meningitis} \sqsubseteq \mathsf{ViralDisease} \end{array} \right\}$$

□

We now have a ranking for our original defeasible KB $\mathcal{K}$ (see Example 1). Once this ranking is identified, then given a query, the core prototypical reasoning algorithm (see Algorithm 1) can be executed to determine if this query follows from the original defeasible KB. $\mathcal{D}_{max}$ represents the infinite rank which contains the classical (non-defeasible) statements from the KB.

Algorithm 1: $Prototypical\ reasoning$

Input: The ranking $\mathcal{D}$ for some KB, $\mathcal{K}$ and a query φ of the form $C \tilde{\sqsubseteq} D$ (or $C \sqsubseteq D$)
Output: **true** if $\mathcal{K} \models_{Prot} C \tilde{\sqsubseteq} D$ (or $\mathcal{K} \models_{Prot} C \sqsubseteq D$), **false** otherwise

1 $n := 1$;
2 **if** $\varphi = C \sqsubseteq D$ **then**
3 **return** $\mathcal{D}_{max} \models C \sqsubseteq D$;
4 **else if** $\varphi = C \tilde{\sqsubseteq} D$ **then**
5 **while** $\bigcup \mathcal{D} \models C \sqsubseteq \bot$ **and** $\mathcal{D} \supset \{\mathcal{D}_{max}\}$ **do**
6 $\mathcal{D} := \mathcal{D} \backslash \{\mathcal{D}_n\}$;
7 $n := n + 1$;
8 **return** $\bigcup \mathcal{D} \models C \sqsubseteq D$;

We now give an example to illustrate the prototypical reasoning algorithm.

Example 3. Consider the ranking from Example 2:

$$\mathcal{D}_{max} = \left\{ \begin{array}{l} \mathsf{BacterialMeningitis} \sqcap \neg\mathsf{Meningitis} \sqsubseteq \bot, \\ \mathsf{ViralMeningitis} \sqcap \neg\mathsf{Meningitis} \sqsubseteq \bot \end{array} \right\}$$

$$\mathcal{D}_2 = \left\{ \begin{array}{l} \mathsf{BacterialMeningitis} \sqsubseteq \mathsf{FatalInfection} \end{array} \right\}$$

$$\mathcal{D}_1 = \left\{ \begin{array}{l} \mathsf{Meningitis} \sqsubseteq \neg\mathsf{FatalInfection}, \\ \mathsf{Meningitis} \sqsubseteq \mathsf{ViralDisease} \end{array} \right\}$$

If we compute prototypical reasoning (via Algorithm 1) for the following query φ = $\mathsf{BacterialMeningitis} \mathrel{\tilde{\sqsubseteq}} \mathsf{ViralDisease}$ then we get the negative result $\mathcal{K} \not\models_{Prot} \varphi$. We find the following motivation for this: We have to execute the else clause of the algorithm because φ is defeasible. The condition on Line 5 of the algorithm holds because $\mathsf{BacterialMeningitis} \sqsubseteq \bot$ is entailed by the set $\mathcal{D}_{max} \cup \mathcal{D}_2 \cup \mathcal{D}_1$. We can derive this because of the following axioms: the first axiom from $\mathcal{D}_{max}$, the only axiom in $\mathcal{D}_2$ and the first axiom in $\mathcal{D}_1$. Therefore, we execute the loop body and prune away the most general information we can. This is rank 1. We execute the check again with the pruned ranking and we find that $\bigcup \mathcal{D} \not\models \mathsf{BacterialMeningitis} \sqsubseteq \bot$. The loop terminates and then we execute Line 8 of the algorithm, i.e., the classical entailment check $\bigcup \mathcal{D} \models \mathsf{BacterialMeningitis} \sqsubseteq \mathsf{ViralDisease}$. We find that this is not the case because $\mathcal{D}_2 \cup \mathcal{D}_{max} \not\models \mathsf{BacterialMeningitis} \sqsubseteq \mathsf{ViralDisease}$. The algorithm then terminates with the negative result.

We can describe the general behaviour of Algorithm 1 by noticing that it works by trying to find the biggest subset of the KB in which the antecedent of our query is *not* exceptional. Intuitively, this can be thought of as finding a typical situation in which the antecedent is satisfied. If the consequent holds in this situation (Line 8) we can conclude that the query follows. □

In the next section we introduce another algorithm for defeasible reasoning that is less conservative (intuitively, wants to make more inferences) than prototypical reasoning. This approach is known as presumptive reasoning.

3.2 Presumptive Reasoning

The presumptive reasoning algorithm is a venturous extension of the prototypical one. Essentially the difference between them (from an algorithmic perspective) is that presumptive reasoning computes an extended version of the ranking that prototypical reasoning does. Presumptive reasoning has a semantics for the propositional case [18] which we shall not discuss here due to space constraints. Intuitively, presumptive reasoning is more lenient than prototypical reasoning in allowing sentences to follow from a defeasible KB, i.e., it *presumes* that some sentence follows from the KB as long as there is no evidence it can find to the contrary. Both algorithms are virtually the same barring one difference: the ranking of sentences in the input KB is computed slightly differently in presumptive reasoning. Because a presumptive ranking is an extension of a prototypical ranking, we describe a procedure for converting a prototypical ranking for a defeasible KB into a presumptive one.

We let $\mathcal{D}$ be a prototypical ranking for some defeasible KB $\mathcal{K}$. Each element $\mathcal{D}_i \in \mathcal{D}$ (which we refer to as a *rank*) is a set of sentences from $\mathcal{K}$. To convert $\mathcal{D}$ to a presumptive ranking $\mathcal{D}'$ we add $|\mathcal{D}_i| - 1$ ranks *above* each $\mathcal{D}_i$ in $\mathcal{D}$. We give an example:

Example 4. Let $\mathcal{D}_i = \{\varphi_1, \varphi_2, \varphi_3\}$ be the only rank in a prototypical ranking $\mathcal{D}$: We recall that in order to extend $\mathcal{D}$ to a presumptive ranking we need to add $|\mathcal{D}_i| - 1$ ranks above each $\mathcal{D}_i$ in $\mathcal{D}$. In Example 4, we have just one rank $\mathcal{D}_i$ with *three* sentences as defined above. We thus need to add *two* ranks above $\mathcal{D}_i$ in $\mathcal{D}$. To understand what these two 'presumptive' ranks will look like we have to explain how a prototypical ranking is used in the prototypical algorithm.

Recall that Lines 5-7 of Algorithm 1 essentially finds a maximal subset of the axioms in $\mathcal{D}$ in which the antecedent of the query is satisfiable. However, the prototypical algorithm does this in a 'clunky' way by removing *entire* ranks at a time (Line 6). Presumptive reasoning is more thorough in finding a maximal subset of the axioms in $\mathcal{D}$ because it only wants to remove the entire rank as a last resort. So it first tries removing (all combinations of) *one* axiom from the lowest rank. If this does not make the antecedent satisfiable then it tries to remove (all combinations of) *two* axioms etc., until it has no other choice but to remove the entire rank. There is an elegant way to capture this behaviour in the presumptive reasoning algorithm without having to naïvely compute all these combinations of sets of axioms. We illustrate this method here for the example ranking $\mathcal{D}$.

We start with the initial rank $\mathcal{D}_i = \{\varphi_1, \varphi_2, \varphi_3\}$. We want to only remove $\mathcal{D}_i$ from $\mathcal{D}$ as a last resort but before that we want to try removing all combinations of one axiom and then two. We use the fact that *removing* one axiom is the same as *keeping* two in a set of three axioms as in Example 4. In other words we would like to compute all unique combinations of *two* axioms holding simultaneously in our $\mathcal{D}_i$. These are: (φ_1 **and** φ_2), (φ_1 **and** φ_3) and (φ_2 **and** φ_3) in Example 4. And this in turn means we have to remove all combinations of one axiom from the KB. These are: $\{\varphi_1\}$, $\{\varphi_2\}$ and $\{\varphi_3\}$ in Example 4. We can implement presumptive reasoning by removing one of these sets from the KB and then performing the check on Line 5 of Algorithm 1. If the antecedent is still unsatisfiable then we put back this set and remove the next one etc. We can see that this exhaustive approach will (in the worst case) require three entailment checks for our example (in general n, for n combinations/sets). Since these entailment checks are computationally quite complex we would like to minimise them as far as possible.

It turns out that there is a way to reduce this operation to a single check as follows: Given the classical counterpart $\mathcal{K}^{\sqsubseteq}$ of a defeasible KB $\mathcal{K}$, let $\mathcal{S}$ = $\{\{\varphi_1, \varphi_2\}, \{\varphi_1, \varphi_3\}, \{\varphi_2, \varphi_3\}\}$ and a let α be a classical subsumption statement, if we want to know whether $\mathcal{K}^{\sqsubseteq} \backslash \{\mathcal{S}_i\} \models \alpha$ for *some* $\mathcal{S}_i \in \mathcal{S}$ where $|\mathcal{S}| = 3$, we do *not* have to perform *all* the checks $\mathcal{K}^{\sqsubseteq} \backslash \{\mathcal{S}_i\} \models \varphi$ where $1 \leq i \leq 3$. We can instead transform the check into something like $\mathcal{K}^{\sqsubseteq} \backslash \{\mathcal{S}_1\}$ **or** $\ldots$ **or** $\mathcal{K}^{\sqsubseteq} \backslash \{\mathcal{S}_3\} \models \alpha$ where this check should have the intuitive meaning: "does at least one $\mathcal{S}_i \in \mathcal{S}$ satisfy $\mathcal{K}^{\sqsubseteq} \backslash \{\mathcal{S}_i\} \models \alpha$?" The question that arises is how do we represent this check in DLs? That is, how do we combine axioms disjunctively or conjunctively? The answer is that there is a result which proves the following rules for DL axioms (and propositional formulas):

$\mathbf{and}(C_i \sqsubseteq D_i)_{i=1}^n = \top \sqsubseteq \sqcap_{i=1}^n (\neg C_i \sqcup D_i)$ and,
$\mathbf{or}(C_i \sqsubseteq D_i)_{i=1}^n = \sqcap_{i=1}^n C_i \sqsubseteq \sqcup_{i=1}^n D_i$

We give the intuition of these rules by noting that: $\mathcal{K}^{\sqsubseteq} \models \mathbf{and}(C_i \sqsubseteq D_i)_{i=1}^n$ iff $\mathcal{K}^{\sqsubseteq} \models C_i \sqsubseteq D_i$ for *all* i such that $1 \leq i \leq n$. Similarly, $\mathcal{K}^{\sqsubseteq} \models \mathbf{or}(C_i \sqsubseteq D_i)_{i=1}^n$ iff $\mathcal{K}^{\sqsubseteq} \models C_i \sqsubseteq D_i$ for *some* i such that $1 \leq i \leq n$.

Returning to our example, removing all combinations of one axiom (keeping two) translates to the following check:

$$\mathcal{K}^{\sqsubseteq} \backslash \{\mathbf{or}(\varphi_1, \varphi_2, \varphi_3)\} \models \alpha$$

The above check can be read as "Is it the case that if we remove at least one set of $\{\{\varphi_1\}, \{\varphi_2\}, \{\varphi_3\}\}$ from $\mathcal{K}^{\sqsubseteq}$ then we can conclude α?". The final iteration for our example is to remove for all combinations of two axioms (keeping one):

$$\mathcal{K}^{\sqsubseteq} \backslash \{\mathbf{or}(\mathbf{and}(\varphi_1, \varphi_2), \mathbf{and}(\varphi_1, \varphi_3), \mathbf{and}(\varphi_2, \varphi_3))\} \models \alpha$$

The above check can be read as "Is it the case that if we remove at least one set of $\{\{\varphi_1, \varphi_2\}, \{\varphi_1, \varphi_3\}, \{\varphi_2, \varphi_3\}\}$ from $\mathcal{K}^{\sqsubseteq}$ then we can conclude α?". We stop at this point because it is clear that the next step is to remove all combinations of three axioms which means we have reached the last resort of removing the entire rank $\mathcal{D}_i$.

The final presumptive conversion $\mathcal{D'}_i$ for the prototypical rank $\mathcal{D}_i$ in Example 4 is $\{\{\varphi_1, \varphi_2, \varphi_3\}, \{\mathbf{or}(\mathbf{and}(\varphi_1, \varphi_2), \mathbf{and}(\varphi_1, \varphi_3), \mathbf{and}(\varphi_2, \varphi_3))\}, \{\mathbf{or}(\varphi_1, \varphi_2, \varphi_3)\}\}$. We can perform the aforementioned transformation on each $\mathcal{D}_i$ in any prototypical ranking to get a corresponding $\mathcal{D'}_i$ and the final presumptive ranking $\mathcal{D'}$ is then computed as $\bigcup \mathcal{D'}_i$. Note that the special rank $\mathcal{D}_{max}$ is not converted using the described procedure but remains as it is because our reasoning algorithms do not manipulate this rank. □

Therefore, to compute presumptive reasoning, we do the following: Obtain a prototypical ranking for the defeasible KB, convert this ranking to a presumptive ranking and then execute the prototypical reasoning algorithm (Algorithm 1) with the presumptive ranking as input. If the result is positive for the algorithm then the query follows *presumptively* from the KB, otherwise the query does *not* follow presumptively. We give a more concrete example of presumptive reasoning below:

Example 5. Let $\mathcal{K}$ be the defeasible KB in Example 1, and $\mathcal{D}$ be the prototypical ranking of its classical counterpart (see Example 2). Also, let our query be the same as in Example 3. $\mathcal{D}$ is denoted as follows:

$$\mathcal{D}_{max} = \left\{ \begin{array}{l} \mathsf{BacterialMeningitis} \sqcap \neg \mathsf{Meningitis} \sqsubseteq \bot, \\ \mathsf{ViralMeningitis} \sqcap \neg \mathsf{Meningitis} \sqsubseteq \bot \end{array} \right\}$$

$$\mathcal{D}_2 = \left\{ \mathsf{BacterialMeningitis} \sqsubseteq \mathsf{FatalInfection} \right\}$$

$$\mathcal{D}_1 = \left\{ \begin{array}{l} \mathsf{Meningitis} \sqsubseteq \neg \mathsf{FatalInfection}, \\ \mathsf{Meningitis} \sqsubseteq \mathsf{ViralDisease} \end{array} \right\}$$

We would like to check if the query $\mathsf{BacterialMeningitis} \sqsubseteq \mathsf{ViralDisease}$ follows presumptively from $\mathcal{K}$ (written as $\mathcal{K} \models_{pres} \mathsf{BacterialMeningitis} \sqsubseteq \mathsf{ViralDisease}$). We start by converting the existing prototypical ranking to a presumptive one by following the procedure described in Example 4. The converted ranking $\mathcal{D'}$ is as follows:

$$\mathcal{D'}_{max} = \left\{ \begin{array}{l} \mathsf{BacterialMeningitis} \sqcap \neg \mathsf{Meningitis} \sqsubseteq \bot, \\ \mathsf{ViralMeningitis} \sqcap \neg \mathsf{Meningitis} \sqsubseteq \bot \end{array} \right\}$$

$$\mathcal{D'}_2 = \left\{ \mathsf{BacterialMeningitis} \sqsubseteq \mathsf{FatalInfection} \right\}$$

$$\mathcal{D'}_{12} = \left\{ \mathsf{Meningitis} \sqsubseteq \neg \mathsf{FatalInfection} \sqcup \mathsf{ViralDisease} \right\}$$

$$\mathcal{D'}_{11} = \left\{ \begin{array}{l} \mathsf{Meningitis} \sqsubseteq \neg \mathsf{FatalInfection}, \\ \mathsf{Meningitis} \sqsubseteq \mathsf{ViralDisease} \end{array} \right\}$$

To compute presumptive reasoning for the query we can use Algorithm 1 with the above presumptive ranking. In this example Algorithm 1 works by trying to find the largest subset of $\mathcal{D'}$ such that the antecedent concept of the query (BacterialMeningitis) is satisfiable (meaning that $\mathsf{BacterialMeningitis} \not\sqsubseteq \bot$ with respect to this subset). In our example, the first presumptive rank $\mathcal{D'}_{11}$ is discarded first and we find that BacterialMeningitis is satisfiable with respect to $\mathcal{D'}_{max} \cup \mathcal{D'}_2 \cup \mathcal{D'}_{12}$. We can therefore stop discarding ranks and perform the check $\mathcal{D'}_{max} \cup \mathcal{D'}_2 \cup \mathcal{D'}_{12} \models \mathsf{BacterialMeningitis} \sqsubseteq \mathsf{ViralDisease}$. In our example, this check holds and we can therefore conclude that $\mathcal{K} \models_{pres} \mathsf{BacterialMeningitis} \sqsubseteq \mathsf{ViralDisease}$. Note the difference in the results between presumptive and prototypical reasoning given the same query. □

It is important to note that other than the difference in the computation of the ranking, the prototypical and presumptive reasoning algorithms are identical. Therefore Algorithm 1 can be executed with a presumptive ranking as input to compute presumptive reasoning for a given defeasible KB and query. We have developed a Protégé plug-in that implements both prototypical and presumptive reasoning for DL-based (and therefore OWL) ontologies. Before we present this plug-in we give a brief introduction to OWL and Protégé.

4. OWL & Protégé

In this section we give an introduction to the OWL family of ontology languages since Protégé is primarily designed to edit and manage OWL ontologies. Thereafter, we give some background on Protégé as well as a brief primer on the anatomy of a Protégé plug-in.

4.1 OWL

Since the advent of the notion of Semantic Web [4] an important task towards achieving this goal has been to develop an appropriate language for constructing ontologies (the building blocks of the Semantic Web). The World Wide Web Consortium (W3C) formed the Web Ontology Working Group to develop such a suitable language. They eventually came up with the Web Ontology Language (OWL) which became a W3C recommendation in 2004 [20]. This initial version is known as OWL or OWL 1. The latest standard OWL 2 superseded OWL 2 as the W3C recommendation in 2009 [27].

The OWL 2 standard is actually a specification of several sublanguages. The most prominent of these are OWL 2 RL, OWL 2 QL, OWL 2 EL, OWL 2 DL and OWL 2 Full. OWL 2 EL is the least expressive of the family but allows for much simpler reasoning. This "species" of OWL was chosen to represent the current versions of important large-scale ontologies in the bio-medical domain such as SNOMED [10], The Gene Ontology (GO) [1] and The National Cancer Institute (NCI) thesaurus [12]. OWL 2 DL is more expressive than OWL 2 EL but less so than OWL 2 Full which is the most expressive OWL language but, as a result, reasoning in this case is undecidable.

OWL ontologies are serialised in various syntaxes such as the Manchester OWL Syntax, OWL/XML and Turtle. The DL axiom $\mathsf{BacterialMeningitis} \sqsubseteq \mathsf{ViralDisease}$, for example, is represented in Figure 1 using OWL/XML syntax:

```
<SubClassOf>
  <Class IRI="#BacterialMeningitis"/>
  <Class IRI="#ViralDisease"/>
</SubClassOf>
```

Figure 1: Example OWL syntax

4.2 Protégé

Protégé (see Figure 2) is an ontology editor developed by the Centre for Biomedical Informatics Research at Stanford University. Older versions of Protégé built ontologies using the frame-based system. The latest version of the editor (Protégé 4.1) is capable of handling ontologies of various formats but predominantly caters for OWL 2 ontologies.

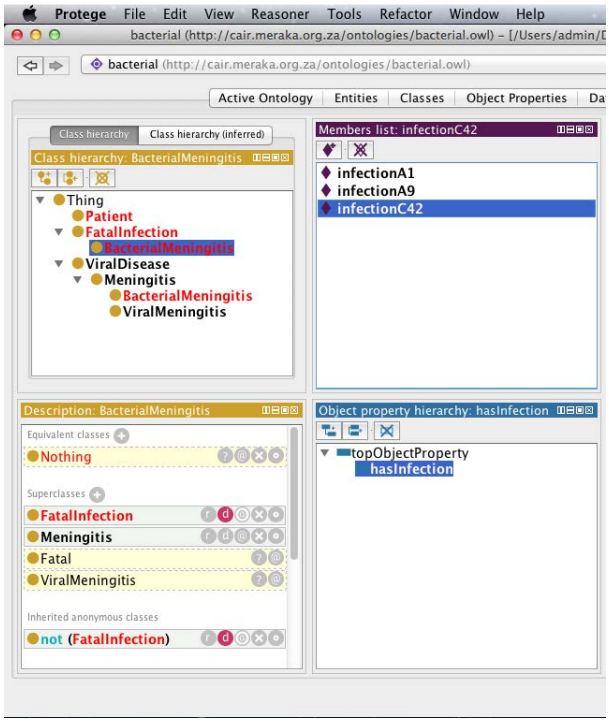

Figure 2: Protégé 4 ontology editor.

The support and alignment with the OWL 2 standard is provided by the latest version of the underlying API that Protégé uses. This is the java-based OWLAPI [11, 14] which it depends on for executing management tasks such as creating, loading and manipulating OWL ontologies. It also provides support for integration of implemented reasoning engines (OWL reasoners) for analysing and revealing implicit information in loaded ontologies during the modelling process. Lastly, it has a plug-in friendly infrastructure which makes it ideal for extensibility. There are many Protégé plug-ins that have been developed by the Knowledge Representation community of varying types and functionality which aim to streamline the ontology engineering process.

4.3 Protégé plug-ins

Protégé plug-ins are categorised by their topic and type.[2] The topic of the plug-in explains the specific service which the plug-in provides. For example, the topics of Protégé can range from simple user interface tweaks to more involved ontology browsing/maintenance tools. Examples are Ontology visualization tools which provide a non-standard visual representation of the ontology (usually to highlight relationships between defined entities in the ontology); Ontology debugging and repair tools [3, 15, 21] which provide services for diagnosing and repairing modelling errors and unwanted consequences in the ontology; And ontology querying tools which allow one to query the ontology for objects with specific properties or characteristics.

The type of plug-in denotes how it is integrated into the Protégé editor itself. The workspace tab plug-ins, as the name suggests, adds their functionality to new tabs in the Protégé interface. This is the case for plug-ins such as the DL Query Tab[3] and our *RaMP* plug-in. A view component plug-in allows one to use or extend the default ontology views, provided by Protégé, in the plug-in implementation. For example, one can write a view component plug-in to display the classes in the ontology in a "tabbed" arrangement (i.e., the classes of the ontology represented in a list with the subclasses indented with respect to their superclasses). There are other types as well such as backend plug-ins, reasoner plug-ins etc.

Protégé has a "plug-in friendly" or modular architecture. In fact, Protégé is a set of interdependent plug-ins which together provide the functionality of the editor. The first step to creating your own Protégé plug-in is to create a build of Protégé in your Integrated Development Environment (IDE) of choice. The recommended way to do this is compile and run Protégé in your IDE from sources in its subversion repository. Detailed instructions on how to do this are available on the Protégé developer documentation pages.[4] Once this is done one can add a new plug-in to the build by initiating a new plug-in project in the IDE. A typical plug-in has four main components:

- Java source code (stored in an appropriate folder structure)
- Manifest file (specifies meta information about plug-in)
- `build.xml` file (specifies instructions on how the plug-in is compiled and run)
- `plug-in.xml` file (specifies further meta information about the plug-in and exactly which part of Protégé it extends)

Detailed instructions on these components and how to compile them in a plug-in are given on the Protégé developer pages.[5]

5. RAMP

We have developed a plug-in for Protégé which allows the ontology modeller to represent defeasible information in the ontology without explicitly extending the underlying ontology language (OWL). This is possible through axiom annotations in Protégé which do not affect the logical meaning of the ontology. Furthermore, this plug-in implements the prototypical and presumptive reasoning algorithms. The plug-in is called *RaMP*[6] which stands for Rational Monotonicity Plug-in. Figure 3 depicts the control panel of RaMP's interface in Protégé 4.1.

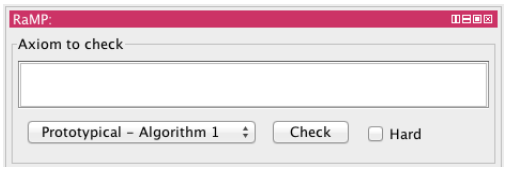

Figure 3: RaMP: Reasoning controller panel

This component of the RaMP interface is called the reasoning controller panel. This panel provides a text input field for the user to enter an axiom (defeasible or hard). The axiom can be verified to follow (or not follow) from the ontology, based on the currently selected reasoning algorithm. This task is accomplished by clicking the "check" button in the same panel. The reasoning algorithm can be selected from a drop-down menu. Currently, only prototypical and presumptive reasoning algorithms are available. There is also a checkbox for indicating if the axiom entered should be treated as defeasible or classical (hard).

[2] `http://protege.stanford.edu/doc/pdk/plug-ins/overview.html`
[3] `http://protegewiki.stanford.edu/wiki/DLQueryTab`
[4] `http://protegewiki.stanford.edu/wiki/Protege4DevDocs`
[5] `http://protegewiki.stanford.edu/wiki/PluginAnatomy`
[6] `http://code.google.com/p/nomor/`

For convenience, RaMP provides a window display (Figure 4) in Protégé to show the user the set of all axioms in the loaded ontology that he/she has currently asserted to be defeasible.

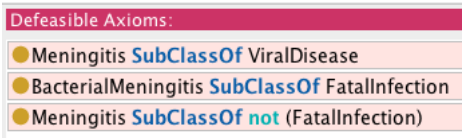

Figure 4: RaMP: Defeasible axioms display

This brings us to the topic of how to assert that an axiom is defeasible in the loaded ontology. RaMP indicates axioms as defeasible in the ontology through a "flagging" mechanism. In the Class Description window in Protégé, the superclasses and equivalent classes of the selected class are indicated. Next to each of these classes there is a button, labelled "d" for defeasibility, which can be clicked to toggle whether that axiom should be viewed as defeasible or not by RaMP. When an axiom is indicated as defeasible the button turns pink and an axiom *annotation* is added to the ontology which stores this information. This mechanism is indicated in Figure 5.

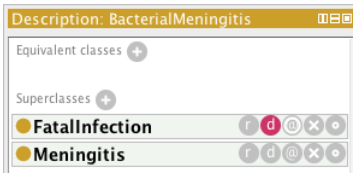

Figure 5: RaMP: Toggling defeasibility of axioms

The reasoning algorithms implemented by RaMP work in a similar way. They compute a ranking (ordering) of the axioms in the ontology. This ranking (see Definition 2) indicates which axioms are more "important" than others in the ontology viewed from the perspective of the algorithm. For formal details of what we mean by important we refer the reader to Definition 3 where the principle of exceptionality is defined. The details of our algorithms are provided in Sections 3.1 and 3.2 respectively. An example ranking as displayed in RaMP is depicted below in Figure 6. The axioms appearing on the light pink background are defeasible axioms while the axioms displayed on the light green background are hard.

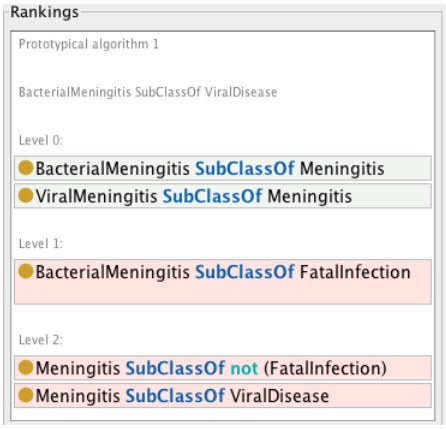

Figure 6: RaMP: Axiom ranking display

As an added feature of RaMP, we have also included a rudimentary facility for the user to fine-tune the computed ranking for the ontology. Adjacent to the defeasible toggle switch in Protégé, there is a button labelled "r" to prompt the user to enter a numerical value representing the rank of the selected axiom. By default, our algorithms compute a ranking of the axioms in the ontology. Each rank of this ranking is a set of axioms. The axioms in a particular rank share the same level of importance according to the algorithm.

What this ranking feature is designed to provide is the freedom for the user to modify the relative importance of the sentences *within* each individual rank. The user is not however allowed to modify the overall ranking structure. This ensures that the reasoning results conform to Rational Closure (see Section 2.2 and Section 3 where we mention this notion). The specific numerical value chosen for the rank of a particular axiom does not matter. What matters is the relative ordering between these values. If axiom φ has rank 1 and it is necessary for axiom φ' to have a higher preference or importance than φ then φ' can have any rank value as long as it is larger than 1. The axiom ranking feature is depicted in Figure 7.

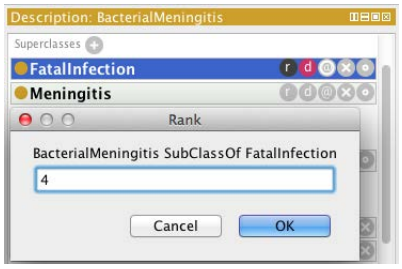

Figure 7: RaMP: Axiom ranking feature

RaMP displays a positive result if the entered axiom follows from the ontology with respect to the selected reasoning algorithm (see Figure 8). In our example, "BacterialMeningitis SubClassOf ViralDisease" follows from the ontology using *presumptive* reasoning:

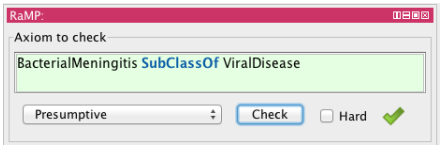

Figure 8: RaMP: Positive result

Conversely, RaMP displays a negative result if the entered axiom does *not* follow from the ontology with respect to the selected reasoning algorithm (see Figure 9). In our example, "BacterialMeningitis SubClassOf ViralDisease" does *not* follow from the ontology using *prototypical* reasoning:

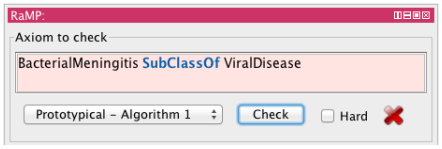

Figure 9: RaMP: Negative result

6. CONCLUSION

We have presented a description of a Protégé plug-in which implements a preliminary version of non-monotonic reasoning for DL-based ontologies. The plug-in provides a mechanism for

indicating defeasible information in the ontology as well as an implementation of two defeasible reasoning algorithms adapted from the work of Lehmann and colleagues [19] in a propositional setting. The plug-in is still in the early stages of development. We would like to introduce a more appropriate interface to facilitate better integration with Protégé, we are also investigating potential optimisations for the defeasible reasoning algorithms. In the future we also plan to include: (*i*) ABox reasoning; (*ii*) Defeasibility in other OWL constructs such as disjointness, roles and role properties; (*iii*) More sophisticated reasoning tasks available in standard monotonic systems such as Classification, Instance checking etc [2] and (*iv*) Other versions of defeasible reasoning (in addition to the ones discussed here). Currently we are evaluating the results reported by the algorithms implemented in the tool. The results will be evaluated in various application domains to identify where each algorithm is most suitable as a reasoning tool.

Acknowledgements

The financial assistance of the National Research Foundation (NRF) towards this research is hereby acknowledged. Opinions expressed and conclusions arrived at, are those of the author and are not necessarily to be attributed to the NRF. The work of Ivan Varzinczak was also supported by the National Research Foundation under Grant number 81225.

7. REFERENCES

[1] M. Ashburner, C. A. Ball, J. A. Blake, D. Botstein, H. Butler, J. M. Cherry, A. P. Davis, K. Dolinski, S. S. Dwight, J. T. Eppig, M. A. Harris, D. P. Hill, L. Issel-Tarver, A. Kasarskis, S. Lewis, J. C. Matese, J. E. Richardson, M. Ringwald, G. M. Rubin, and G. Sherlock. Gene Ontology: tool for the unification of biology. *Nature Genetics*, 25:25–29, 2000.

[2] F. Baader, D. Calvanese, D. McGuinness, D. Nardi, and P. Patel-Schneider, editors. *The description logic handbook*. Cambridge, 2 edition, 2007.

[3] F. Baader and R. Peñaloza. Axiom pinpointing in general tableaux. In *Automated Reasoning with Analytic Tableaux and Related Methods*, volume 4548 of *Lecture Notes in Computer Science*, pages 11–27. Springer Berlin / Heidelberg, 2007.

[4] T. Berners-Lee, J. Hendler, and O. Lassila. The Semantic Web. In *Scientific American*. Scientific American, May, 2001.

[5] R. J. Brachman and H. J. Levesque. The tractability of subsumption in frame-based description languages. In *Proceedings of the Third National Conference on Artificial Intelligence*, pages 34–37. American Association for Artificial Intelligence, 1984.

[6] K. Britz, J. Heidema, and T. Meyer. Semantic preferential subsumption. In *Proc. of KR*, pages 476–484, 2008.

[7] K. Britz, T. Meyer, and I. Varzinczak. Preferential reasoning for modal logics. *Electronic Notes in Theoretical Computer Science*, 278:55–69, 2011. Proc. of the Workshop on Methods for Modalities.

[8] K. Britz, T. Meyer, and I. Varzinczak. Semantic foundation for preferential description logics. In *Proc. of the Australasian Conference on Artificial Intelligence*, 2011.

[9] G. Casini and U. Straccia. Rational closure for defeasible description logics. In *Proc. of JELIA*, pages 77–90, 2010.

[10] R. A. Cotè, editor. *SNOMED International: The systematized nomenclature of human and veterinary medicine*. College of American Pathologists, 3rd edition, 1993.

[11] B. Cuenca-Grau, I. Horrocks, B. Motik, B. Parsia, P. F. Patel-Schneider, and U. Sattler. Cooking the semantic web with the OWL API. *Web Semantics: Science, Services and Agents on the World Wide Web*, 6(4):309–322, 2008.

[12] S. de Coronado, M. W. Haber, N. Sioutos, M. S. Tuttle, and L. W. Wright. NCI Thesaurus: using science-based terminology to integrate cancer research results. *Medinfo*, 11:33–37, 2004.

[13] L. Giordano, N. Olivetti, V. Gliozzi, and G. Pozzato. $\mathcal{ALC} + T$: A preferential extension of description logics. *Fundamenta Informaticae*, 96(3):341–372, 2009.

[14] M. Horridge and S. Bechhofer. The OWL API: A Java API for OWL ontologies. *Semantic Web Journal*, pages 1–11, 2010. `http://www.semantic-web-journal.net/sites/default/files/swj107_2.pdf`.

[15] M. Horridge, B. Parsia, and U. Sattler. Explaining inconsistencies in owl ontologies. In *Scalable Uncertainty Management*, volume 5785 of *Lecture Notes in Computer Science*, pages 124–137. Springer Berlin / Heidelberg, 2009.

[16] H. Knublauch, R. W. Fergerson, N. F. Noy, and M. A. Musen. The Protégé OWL Plugin: An Open Development Environment for Semantic Web Applications. In *Proceedings of the Third International Semantic Web Conference (ISWC)*, volume 3298 of *Lecture Notes in Computer Science*, 2004.

[17] S. Kraus, D. Lehmann, and M. Magidor. Nonmonotonic reasoning, preferential models and cumulative logics. *Artificial Intelligence*, 44:167–207, 1990.

[18] D. Lehmann. Another perspective on default reasoning. *Annals of Mathematics and Artificial Intelligence*, 15:61–82, 1995.

[19] D. Lehmann and M. Magidor. What does a conditional knowledge base entail? *Artificial Intelligence*, 55(1):1–60, 1992.

[20] D. L. Mcguinness and F. van Harmelen. OWL Web Ontology Language Overview, Feb. 2004. `http://www.w3.org/TR/owl-features`.

[21] K. Moodley. Debugging and repair of Description Logic ontologies. Master's thesis, University of KwaZulu-Natal, 2011.

[22] J. Quantz. A preference semantics for defaults in terminological logics. In *Proc. of KR*, pages 294–305, 1992.

[23] K. Schild. A correspondence theory for terminological logics: Preliminary report. In *Proceedings of the 12th International Joint Conference on Artificial Intelligence (IJCAI)*, pages 466–471, 1991.

[24] R. Shearer, B. Motik, and I. Horrocks. HermiT: a highly efficient OWL reasoner. In *Proceedings of the Fifth International Workshop on OWL: Experiences and Directions (OWLED)*, 2008.

[25] E. Sirin, B. Parsia, B. C. Grau, A. Kalyanpur, and Y. Katz. Pellet: A practical OWL-DL reasoner. *Web Semantics: Science, Services and Agents on the World Wide Web*, 5(2):51 – 53, 2007.

[26] D. Tsarkov and I. Horrocks. FaCT++ description logic reasoner: System description. In *Automated Reasoning*, volume 4130 of *Lecture Notes in Computer Science*, pages 292–297. Springer Berlin / Heidelberg, 2006.

[27] W3C OWL Working Group. OWL 2 Web Ontology Language Document Overview, W3C Recommendation, World Wide Web Consortium, 2009. `http://www.w3.org/TR/owl2-overview`.