A Protégé Plug-in for Defeasible Reasoning

Conference Paper · June 2012

CITATIONS
5

READS
53

3 authors, including:

Kody Moodley
Maastricht University
13 PUBLICATIONS **88** CITATIONS

SEE PROFILE

A Protégé Plug-in for Defeasible Reasoning

Kody Moodley, Thomas Meyer, and Ivan Varzinczak

Centre for Artificial Intelligence Research
CSIR Meraka and University of KwaZulu-Natal, South Africa.
{kmoodley,tmeyer,ivarzinczak}@csir.co.za

Abstract. We discuss two approaches for *defeasible* reasoning in Description Logics that allow for the statement of defeasible subsumptions of the form "α subsumed by β *usually* holds". These 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 the propositional case. Here we recast their definitions in a defeasible DL context and define algorithms for prototypical and presumptive reasoning in defeasible DL knowledge bases. In particular, we present a plug-in for the Protégé ontology editor which implements these algorithms for OWL ontologies. The plug-in is called *RaMP* and allows the modeller to indicate defeasible information in OWL ontologies and check entailment of defeasible subsumptions from defeasible knowledge bases.

1 Introduction

Entailment in standard DL reasoning systems is *monotonic*. Monotonicity is a property of knowledge representation (KR) systems that are built upon classical logics. It specifies that knowledge is always 'incremental'. That is, adding to (or strengthening) the information in a Knowledge Base (KB) cannot result in any previously known conclusions being retracted from the KB. In classical logics, 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 $\alpha \sqsubseteq \beta$ it follows that $\alpha \sqcap \gamma \sqsubseteq \beta$ for any γ.

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. Let us consider an example. We select a domain which describes power plants (specifically *nuclear* power plants) and their safety under certain conditions. The following concept and role names will be used: PP (power plants), NPP (nuclear power plants), BrNPP (Brazilian nuclear power plants), $\mathsf{SeismicArea}$ (tracts of land prone to seismic activity), $\mathsf{DangerousPP}$ (power plants which are unsafe due to some conditions) and $\mathsf{isLocIn}$ (a property of an entity indicating where it is located geographically). We specify a simple example using the vocabulary described above. Consider the following DL knowledge base

$$\mathcal{K} = \left\{ \mathsf{BrNPP} \sqsubseteq \mathsf{NPP}, \mathsf{NPP} \sqsubseteq \neg\mathsf{DangerousPP} \right\}$$

From $\mathcal{K}$ we conclude classically that a Brazilian nuclear power plant is *not* a dangerous power plant ($\mathsf{BrNPP} \sqsubseteq \neg\mathsf{DangerousPP}$). This conclusion displays typical monotonic reasoning behaviour. That is, enforcing that *all* Brazilian nuclear power plants are nuclear power plants ($\mathsf{BrNPP} \sqsubseteq \mathsf{NPP}$) and that *all* nuclear power plants are *not* dangerous power plants ($\mathsf{NPP} \sqsubseteq \neg\mathsf{DangerousPP}$), it is also implicitly enforced that *all* Brazilian nuclear power plants are *not* dangerous power plants. □

From an ontology modeller's perspective, the type of reasoning behaviour exhibited in the example may not be suitable always. This is because the modeller may want to be able to cater for exceptions. We may not want to enforce that *all* nuclear power plants are safe. Rather, we may want to represent something less rigid. For example, one may want to express that in the most normal cases nuclear power plants are not dangerous but that there may be some exceptional cases in which they *are* dangerous (*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 systems.

Returning to the example above, we argue that one needs to develop a defeasible reasoning approach to capture intuitions like: In general, nuclear power plants are not dangerous *but* a specific type of nuclear power plant (e.g., a Brazilian nuclear power plant located in a seismic area) may be dangerous. It would be useful to have a robust system in the DL setting that is capable of implementing this kind of reasoning.

The goal of this paper is to present and demonstrate a preliminary version of the software defeasible reasoning system that we have developed to be used with OWL ontologies. This system, known as *RaMP*, has been packaged as a plug-in for the popular Protégé ontology editor. The plug-in is discussed in more detail in Section 4.

2 Background

There are various approaches for introducing non-monotonic reasoning capabilities in logic-based knowledge representation systems. Among these, the approach by Kraus, Lehmann and Magidor (often called the KLM approach) [7,9] has been particularly successful due to its elegance and robustness. They enrich propositional logic with a defeasible 'implication' operator ($\mathrel{|\!\sim}$). This operator allows one to write down *defeasible implication* statements (also called *conditional assertions*) of the form $\alpha \mathrel{|\!\sim} \beta$, where α and β are propositional formulas. The sentence $\alpha \mathrel{|\!\sim} \beta$ 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). Many extensions of the KLM approach have been proposed in the literature recently [10,6,2,5,3,4]. The approach that we use in our system is based on the KLM approach as well. 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 remains to be defined what a valid inference from a conditional KB is.

Lehmann et al. characterize the notion of *rational closure* [9, Section 5] and motivate this notion to be a suitable answer to the above question. The name given to the

reasoning approach that computes rational closure for conditional KBs is *prototypical reasoning*. Lehmann and colleagues also discuss similar forms of reasoning that may be suitable for computing inferences from conditional KBs. One such type of reasoning is known as *presumptive reasoning* [8]. Presumptive reasoning is actually a venturous extension of prototypical reasoning. This means that any inference that follows from a conditional KB using prototypical reasoning will also follow from the KB using presumptive reasoning. The converse is not necessarily true.

Our approach to defeasible reasoning is based on the work by Britz et al. [4] in which they propose a defeasible extension of the DL $\mathcal{ALC}$. The defeasibility is introduced through a *defeasible subsumption* operator ($\sqsubseteq\!\!\!\!\sim$). This operator is 'supraclassical' to the classical subsumption operator ($\sqsubseteq$). The semantics of defeasible subsumption is similar to that for the $\mid\!\sim$ operator given by Lehmann and colleagues. Intuitively the semantics states that given a defeasible subsumption axiom $\alpha \sqsubseteq\!\!\!\!\sim \beta$ (where α and β may be complex $\mathcal{ALC}$ concepts), then this statement means that the most *typical* α's are also β's (as opposed to *all* α's being β's in the classical case). As of writing, only a semantics for defeasible *subsumption* is explicitly provided by the approach (although defeasible *equivalence* follows trivially as well). Britz et al's approach is only applicable to DL TBoxes currently.

3 Prototypical and Presumptive Reasoning

Prototypical and presumptive reasoning are answers to the important question: given a KB that contains defeasible subsumption statements like $\alpha \sqsubseteq\!\!\!\!\sim \beta$, what does it mean for one to draw inferences from this KB and how do we compute these inferences? Lehmann et al. have provided answers by developing the notions of prototypical and presumptive reasoning (albeit in a propositional context) and also specifying algorithms to compute these. It turns out that these notions are adaptable to the DL setting [4] and to this end we have adapted the algorithms by Lehmann et al. [9] for computing both prototypical and presumptive reasoning for a *defeasible KB* (an $\mathcal{ALC}$ KB that may additionally contain defeasible subsumption statements of the form $\alpha \sqsubseteq\!\!\!\!\sim \beta$).

3.1 Prototypical Reasoning

Prototypical reasoning corresponds exactly to the propositional notion of rational closure, lifted to the DL case. Due to space constraints, we are not concerned with the semantics here. Interested readers may consult the work of Lehmann and colleagues [9] for the semantics for rational closure in a propositional context. For a characterization of rational closure for the DL $\mathcal{ALC}$ one can consult the work of Britz et al. [4].

The algorithm for prototypical reasoning, takes as input a subsumption statement (also called a *query*) which may be defeasible or classical and a defeasible KB. The output of the algorithm is **true** if the query φ follows (prototypically) from the given KB $\mathcal{K}$ (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 them (in the eyes 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:

Definition 1 (transformKB). *Given a defeasible KB $\mathcal{K}$:*
$transformKB(\mathcal{K}) := \{transform(\varphi) | \varphi \in \mathcal{K}\}$,
where $transform(\varphi) := \begin{cases} \alpha \sqcap \neg\beta \sqsubseteq \bot, \ if \ \varphi = \alpha \sqsubseteq \beta \\ \alpha \sqsubseteq \beta, \ if \ \varphi = \alpha \tilde{\sqsubseteq} \beta \end{cases}$

Example 1. Let $\mathcal{K}$ be the input of the procedure in Definition 1:

$$\mathcal{K} = \left\{ \begin{array}{l} \mathsf{BrNPP} \sqsubseteq \exists \mathsf{hasFault}, \mathsf{BrNPP} \sqsubseteq \mathsf{NPP}, \\ \mathsf{BrNPP} \sqsubseteq \exists \mathsf{isLocIn.Brazil}, \mathsf{NPP} \tilde{\sqsubseteq} \neg \exists \mathsf{hasFault}, \\ \mathsf{NPP} \sqcap \exists \mathsf{isLocIn.SeismicArea} \tilde{\sqsubseteq} \mathsf{DangerousPP}, \\ \exists \mathsf{hasFault} \tilde{\sqsubseteq} \mathsf{DangerousPP}, \mathsf{DangerousPP} \tilde{\sqsubseteq} \exists \mathsf{isLocIn.SeismicArea} \end{array} \right\}$$

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

$$\mathcal{K}' = \left\{ \begin{array}{l} \mathsf{BrNPP} \sqcap \neg \exists \mathsf{hasFault} \sqsubseteq \bot, \mathsf{BrNPP} \sqcap \neg \mathsf{NPP} \sqsubseteq \bot, \\ \mathsf{BrNPP} \sqcap \neg \exists \mathsf{isLocIn.Brazil} \sqsubseteq \bot, \mathsf{NPP} \sqsubseteq \neg \exists \mathsf{hasFault}, \\ \mathsf{NPP} \sqcap \exists \mathsf{isLocIn.SeismicArea} \sqsubseteq \mathsf{DangerousPP}, \\ \exists \mathsf{hasFault} \sqsubseteq \mathsf{DangerousPP}, \mathsf{DangerousPP} \sqsubseteq \exists \mathsf{isLocIn.SeismicArea} \end{array} \right\}$$

□

The second procedure of the prototypical algorithm is the computation of a *ranking* of sentences in the KB according to the notion of *exceptionality* [9]. 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). *Given a defeasible KB $\mathcal{K}$, a* ranking *for $\mathcal{K}$ is a total preorder on the elements (axioms) in $\mathcal{K}$, with axioms higher up in the ordering interpreted as having a higher preference/importance.*

Note that a ranking can in practice be implemented as a collection where each element of this collection is a set of sentences from the KB. Each sentence in a particular set has the same magnitude of importance (we also call this a *rank*). In the prototypical reasoning algorithm, the ranking of a defeasible KB is computed according to the

notion of exceptionality. Intuitively, a concept α is said to be exceptional w.r.t. a defeasible KB $\mathcal{K}$, if it is the case that $\top \sqsubseteq \neg\alpha$ *usually* follows from $\mathcal{K}$. That is, *typically* everything is in $\neg\alpha$, thereby making α an exception to this rule. In the context of our algorithm, checking whether α is exceptional w.r.t $\mathcal{K}$ can be reduced to checking if $\mathcal{K}'$ *classically* entails $\alpha \sqsubseteq \bot$ where $\mathcal{K}'$ is the classical transformation of $\mathcal{K}$.

Definition 3 (Exceptionality). *Let $\mathcal{K}$ be a defeasible KB and α, β concepts. Then:*

- *α is* exceptional *w.r.t. $\mathcal{K}$ iff $\mathcal{K}' \models \alpha \sqsubseteq \bot$. Where $\mathcal{K}'$ is the transformation of $\mathcal{K}$.*
- *$\alpha \sqsubseteq \beta$ is* exceptional *w.r.t. $\mathcal{K}$ iff α is* exceptional *w.r.t. $\mathcal{K}$.*
- *$\alpha \sqsubseteq \beta$ is* exceptional *w.r.t. $\mathcal{K}$ iff $\alpha \sqcap \neg\beta$ 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 $\mathcal{K}$).*

The above notion of exceptionality is included in the computation of the ranking for a defeasible KB as sub-procedure $exceptional(\mathcal{E}) := \{\alpha \sqsubseteq \beta \in \mathcal{E} \mid \mathcal{E} \models \alpha \sqsubseteq \bot\}$.

Now that we have detailed the sub-procedures and principles needed to describe the computation of a ranking of the sentences in a defeasible KB, we specify the *complete* procedure for implementing this:

Procedure $computeRanking(\mathcal{K})$

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

1 $i := 0$; $\mathcal{K}' :=$ *transformKB*$(\mathcal{K})$; $\mathcal{E}_0 := \mathcal{K}'$; $\mathcal{E}_1 :=$ *exceptional*$(\mathcal{E}_0)$;
2 **while** $\mathcal{E}_{i+1} \neq \mathcal{E}_i$ **do**
3 $\quad i := i + 1$; $\mathcal{E}_{i+1} :=$ *exceptional*$(\mathcal{E}_i)$;
4 $n := i$; $\mathcal{D}_\infty := \mathcal{E}_n$; $\mathcal{D} := \{\mathcal{D}_\infty\}$;
5 **for** $j := 1$ *to* n **do**
6 $\quad \mathcal{D}_j := \mathcal{E}_{j-1} \backslash \mathcal{E}_j$; $\mathcal{D} := \mathcal{D} \cup \{\mathcal{D}_j\}$;
7 **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 element in any of such sets shares the same level of importance in the algorithm.

Example 2. Consider the transformed KB from Example 1:

$$\mathcal{K}' = \left\{ \begin{array}{l} \mathsf{BrNPP} \sqcap \neg\exists\mathsf{hasFault} \sqsubseteq \bot, \mathsf{BrNPP} \sqcap \neg\mathsf{NPP} \sqsubseteq \bot, \\ \mathsf{BrNPP} \sqcap \neg\exists\mathsf{isLocIn.Brazil} \sqsubseteq \bot, \mathsf{NPP} \sqsubseteq \neg\exists\mathsf{hasFault}, \\ \mathsf{NPP} \sqcap \exists\mathsf{isLocIn.SeismicArea} \sqsubseteq \mathsf{DangerousPP}, \\ \exists\mathsf{hasFault} \sqsubseteq \mathsf{DangerousPP}, \mathsf{DangerousPP} \sqsubseteq \exists\mathsf{isLocIn.SeismicArea} \end{array} \right\}$$

$\mathcal{K}'$ (second statement on Line 1 of the $computeRanking$ procedure) is calculated as in Example 1. Lines 2 to 3 of the $computeRanking$ procedure are responsible for computing the different levels of exceptional sentences according to Definition 3. This

segment of the procedure executes Procedure $exceptional$ until $\mathcal{E}_{i+1} = \mathcal{E}_i$ (a fixed point is reached). In the example when Line 1 of $computeRanking$ is executed $\mathcal{E}_0 := \mathcal{K}'$ and $\mathcal{E}_1 := exceptional(\mathcal{E}_0)$. If we do the computation as detailed in $exceptional$ then we find: $\mathcal{E}_1 = \{\mathsf{BrNPP} \sqcap \neg\exists\mathsf{hasFault} \sqsubseteq \bot, \mathsf{BrNPP} \sqcap \neg\mathsf{NPP} \sqsubseteq \bot, \mathsf{BrNPP} \sqcap \neg\exists\mathsf{isLocIn}.\mathsf{Brazil} \sqsubseteq \bot\}$.

We see that $\mathcal{E}_1 \neq \mathcal{E}_0$ therefore we have to execute the while loop again to find $\mathcal{E}_2$. The result of this computation shows us that $\mathcal{E}_2 = \mathcal{E}_1$ and we have thus reached a fixed point. The while loop terminates and the final ranking $\mathcal{D}$ can be computed as specified by Lines 4 to 6 of the procedure. The final ranking $\mathcal{D}$ is composed of:

$$\mathcal{D}_\infty = \left\{ \begin{array}{l} \mathsf{BrNPP} \sqcap \neg\exists\mathsf{hasFault} \sqsubseteq \bot, \mathsf{BrNPP} \sqcap \neg\mathsf{NPP} \sqsubseteq \bot, \\ \mathsf{BrNPP} \sqcap \neg\exists\mathsf{isLocIn}.\mathsf{Brazil} \sqsubseteq \bot \end{array} \right\}$$

$$\mathcal{D}_1 = \left\{ \begin{array}{l} \mathsf{NPP} \sqsubseteq \neg\exists\mathsf{hasFault}, \exists\mathsf{hasFault} \sqsubseteq \mathsf{DangerousPP}, \\ \mathsf{NPP} \sqcap \exists\mathsf{isLocIn}.\mathsf{SeismicArea} \sqsubseteq \mathsf{DangerousPP}, \\ \mathsf{DangerousPP} \sqsubseteq \exists\mathsf{isLocIn}.\mathsf{SeismicArea} \end{array} \right\}$$

□

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. $\mathcal{D}_\infty$ 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 $\alpha \mathrel{|\!\sim} \beta$ (or $\alpha \sqsubseteq \beta$)
Output: **true** if $\mathcal{K} \models_{Prot} \alpha \mathrel{|\!\sim} \beta$ (or $\mathcal{K} \models_{Prot} \alpha \sqsubseteq \beta$), **false** otherwise

1 $n := 1$;
2 **if** $\varphi = \alpha \sqsubseteq \beta$ **then**
3 **return** $\mathcal{D}_\infty \models \alpha \sqsubseteq \beta$;
4 **else if** $\varphi = \alpha \mathrel{|\!\sim} \beta$ **then**
5 **while** $\bigcup \mathcal{D} \models \alpha \sqsubseteq \bot$ **and** $\mathcal{D} \neq \emptyset$ **do**
6 $\mathcal{D} := \mathcal{D} \backslash \{\mathcal{D}_n\}$; $n := n + 1$;
7 **return** $\bigcup \mathcal{D} \models \alpha \sqsubseteq \beta$;

We give an example to illustrate the prototypical reasoning algorithm below:

Example 3. Consider the following defeasible KB $\mathcal{K}$ with six axioms:

$$\mathcal{K} = \left\{ \begin{array}{l} \mathsf{BrNPP} \sqsubseteq \mathsf{NPP}, \mathsf{NPP} \sqsubseteq \mathsf{PP}, \exists\mathsf{isLocIn} \sqsubseteq \mathsf{PP}, \\ \exists\mathsf{isLocIn}.\mathsf{SeismicArea} \sqsubseteq \mathsf{DangerousPP}, \mathsf{NPP} \mathrel{|\!\sim} \neg\mathsf{DangerousPP}, \\ \mathsf{BrNPP} \mathrel{|\!\sim} \exists\mathsf{isLocIn}.\mathsf{SeismicArea} \end{array} \right\}$$

Considering higher indices to represent more exceptional sentences, the ranking for this KB (computed using relevant procedures discussed) is the following:

$$\mathcal{D}_\infty = \left\{ \begin{array}{l} \mathsf{BrNPP} \sqcap \neg \mathsf{NPP} \sqsubseteq \bot, \exists \mathsf{isLocIn} \sqcap \neg \mathsf{PP} \sqsubseteq \bot, \\ \exists \mathsf{isLocIn.SeismicArea} \sqcap \neg \mathsf{DangerousPP} \sqsubseteq \bot, \mathsf{NPP} \sqcap \neg \mathsf{PP} \sqsubseteq \bot \end{array} \right\}$$

$$\mathcal{D}_1 = \{\mathsf{BrNPP} \sqsubseteq \exists \mathsf{isLocIn.SeismicArea}\}$$

$$\mathcal{D}_2 = \{\mathsf{NPP} \sqsubseteq \neg \mathsf{DangerousPP}\}$$

If we compute prototypical reasoning (via Algorithm 1) for the following query $\varphi = \mathsf{BrNPP} \mathrel{\tilde{\sqsubseteq}} \mathsf{DangerousPP}$ then we get the positive result $\mathcal{K} \models_{Prot} \varphi$. We find the following motivation for this: the condition on Line 5 of the algorithm holds because $\mathsf{BrNPP} \sqsubseteq \bot$ follows from $\mathcal{D}$ ($\mathcal{D}_\infty \cup \mathcal{D}_1 \cup \mathcal{D}_2$) and $\mathcal{D} \neq \emptyset$. Therefore we execute the loop body and $\mathcal{D}$ becomes $\mathcal{D}_\infty \cup \mathcal{D}_1$. The loop condition no longer holds because $\bigcup \mathcal{D} \not\models \mathsf{BrNPP} \sqsubseteq \bot$ and therefore the loop terminates.

Last, the classical entailment check ($\bigcup \mathcal{D} \models \mathsf{BrNPP} \sqsubseteq \mathsf{DangerousPP}$) on Line 7 is performed. This returns **true** in our example and therefore $\mathcal{K} \models_{Prot} \varphi$.

An intuitive reading of this result shows us that *typically* nuclear power plants are *not* dangerous power plants ($\mathsf{NPP} \mathrel{\tilde{\sqsubseteq}} \neg \mathsf{DangerousPP}$). If we examine the query φ we see that we are not in a typical situation because we have a *specific type* of nuclear power plant, i.e., A *Brazilian* nuclear power plant. Additionally, we know that typical Brazilian nuclear power plants are located in seismic areas stated by ($\mathsf{BrNPP} \mathrel{\tilde{\sqsubseteq}} \exists \mathsf{isLocIn.SeismicArea}$) and typical power plants located in seismic areas are dangerous ($\exists \mathsf{isLocIn} \sqsubseteq \mathsf{PP}$, $\exists \mathsf{isLocIn.SeismicArea} \sqsubseteq \mathsf{DangerousPP}$). Given this information, prototypical reasoning allows for the possibility that this type of nuclear power plant *is* dangerous, which represents an exception to the general rule that $\mathsf{NPP} \mathrel{\tilde{\sqsubseteq}} \neg \mathsf{DangerousPP}$. □

Next we introduce an alternative defeasible reasoning algorithm to the prototypical approach just described. 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 [8] which we shall not discuss here due to space constraints. The algorithm for computing 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 $\mathcal{D}$.

Example 4. Let $\mathcal{D}_i = \{\varphi_1, \varphi_2, \varphi_3\}$ be a rank in some 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 $\mathcal{D}$. For our example, let us consider that we have a prototypical ranking containing just one rank $\mathcal{D}_i$ 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 5-6 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 first tries removing (all permutations of) *one* axiom from the highest rank, instead of the entire rank. If this does not make the antecedent satisfiable then it tries removing (all permutations of) *two* axioms from the same rank etc. until eventually (all permutations of) $|\mathcal{D}_i| - 1$ axioms are removed. We now show an elegant way to perform this fine-grained removal.

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 permutations of one axiom and then two. Instead of doing this in a naïve way by computing all these permutations, we use the fact that *removing* one axiom is the same as *keeping* two in a set of three axioms as in the example. In other words we would like to compute all instances of *two* axioms holding simultaneously in our $\mathcal{D}_i$. That is $(\varphi_1 \mathbf{and} \varphi_2) \mathbf{or} (\varphi_1 \mathbf{and} \varphi_3) \mathbf{or} (\varphi_2 \mathbf{and} \varphi_3)$ in our example.

It turns out that there is a way to transform this 'check' or expression into a *single axiom* according to the following rules for DL axioms:

$\mathbf{and}(\alpha_i \sqsubseteq \beta_i)_{i=1}^n = \top \sqsubseteq \sqcap_{i=1}^n (\neg\alpha_i \sqcup \beta_i)$ and,
$\mathbf{or}(\alpha_i \sqsubseteq \beta_i)_{i=1}^n = \sqcap_{i=1}^n \alpha_i \sqsubseteq \sqcup_{i=1}^n \beta_i$, where $\alpha_i \sqsubseteq \beta_i = \varphi_i$.

If removing all permutations of one axiom (keeping two) does not make the antecedent satisfiable in our example then we try removing all permutations of *two* (keeping one). Therefore for our example the expression to check to verify this is $\varphi_1 \mathbf{or} \varphi_2 \mathbf{or} \varphi_3$. Therefore the presumptive conversion $\mathcal{D}'_i$ for the prototypical rank $\mathcal{D}_i$ is $\{((\varphi_1 \mathbf{and} \varphi_2) \mathbf{or} (\varphi_1 \mathbf{and} \varphi_3) \mathbf{or} (\varphi_2 \mathbf{and} \varphi_3)), (\varphi_1 \mathbf{or} \varphi_2 \mathbf{or} \varphi_3), \{\varphi_1, \varphi_2, \varphi_3\}\}$. This process is repeated on each $\mathcal{D}_i$ in a prototypical ranking and the final presumptive ranking $\mathcal{D}'$ is computed as $\bigcup \mathcal{D}'_i$. □

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. We present this plug-in in the next section.

4 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*[1] which stands for Rational Monotonicity Plug-in. Figure 1 depicts the control panel of RaMP's interface in Protégé 4.1.

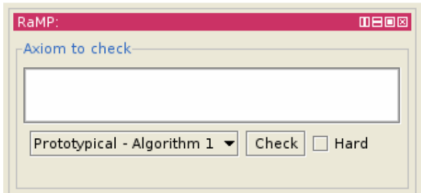

Fig. 1: RaMP: Reasoning controller panel

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 is defeasible or classical. If the axiom is classical, we also refer to it as a *hard* axiom. For convenience, RaMP provides a window display (Figure 2) in Protégé to show the user which axioms in the loaded ontology are indicated as defeasible axioms.

Fig. 2: 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" process. 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 (see Figure 3).

[1] `http://code.google.com/p/nomor/`

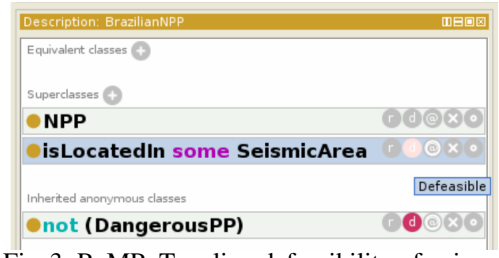

Fig. 3: RaMP: Toggling defeasibility of axioms

The reasoning algorithms implemented by RaMP work in a similar way. They compute a ranking (ordering) of the axioms in the ontology and use this ranking to determine which axioms are more important than others in the ontology. The details of these algorithms will are provided in Sections 3.1 and 3.2 respectively. An example ranking is depicted below in Figure 4. The axioms appearing on the light pink background are defeasible axioms while the others are hard.

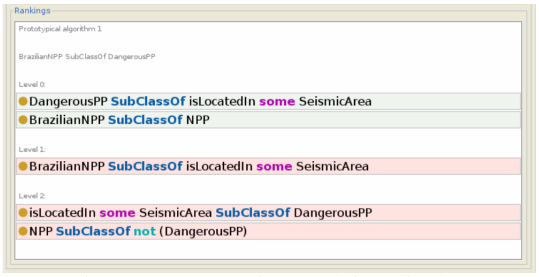

Fig. 4: RaMP: Axiom ranking display

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. The specific numerical value chosen 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 rank then it does not matter whether axiom φ has rank 2 or 200 as long as the rank of φ' is greater than 2 or 200 respectively. The axiom ranking feature is depicted in Figure 5.

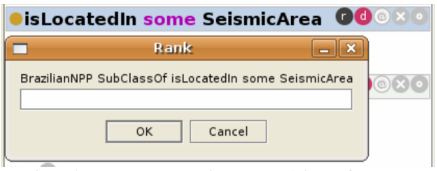

Fig. 5: RaMP: Axiom ranking feature

5 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 [9] in a propositional setting. The plug-in is still in the early stages of development. We would like to introduce a more elaborate interface to facilitate better integration with Protégé, we are also investigating potential optimizations 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 monontonic systems such as Classification, Instance checking etc [1] 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.

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.

References

1. Baader, F., Calvanese, D., McGuinness, D., Nardi, D., Patel-Schneider, P. (eds.): The description logic handbook. Cambridge, 2 edn. (2007)
2. Britz, K., Heidema, J., Meyer, T.: Semantic preferential subsumption. In: Proc. of KR. pp. 476–484 (2008)
3. Britz, K., Meyer, T., Varzinczak, I.: Preferential reasoning for modal logics. Electronic Notes in Theoretical Computer Science 278, 55–69 (2011), proc. of the Workshop on Methods for Modalities
4. Britz, K., Meyer, T., Varzinczak, I.: Semantic foundation for preferential description logics. In: Proc. of the Australasian Conference on Artificial Intelligence (2011)
5. Casini, G., Straccia, U.: Rational closure for defeasible description logics. In: Proc. of JELIA. pp. 77–90 (2010)
6. Giordano, L., Olivetti, N., Gliozzi, V., Pozzato, G.: $\mathcal{ALC} + T$: A preferential extension of description logics. Fundamenta Informaticae 96(3), 341–372 (2009)
7. Kraus, S., Lehmann, D., Magidor, M.: Nonmonotonic reasoning, preferential models and cumulative logics. Artificial Intelligence 44, 167–207 (1990)
8. Lehmann, D.: Another perspective on default reasoning. Annals of Mathematics and Artificial Intelligence 15, 61–82 (1995)
9. Lehmann, D., Magidor, M.: What does a conditional knowledge base entail? Artificial Intelligence 55(1), 1–60 (1992)
10. Quantz, J.: A preference semantics for defaults in terminological logics. In: Proc. of KR. pp. 294–305 (1992)