Visualization of Proofs in Defeasible Logic

Ioannis Avguleas[1,2], Katerina Gkirtzou[1,2], Sofia Triantafilou[1,2], Antonis Bikakis[1,2], Grigoris Antoniou[1,2], Efstratios Kontopoulos[3], and Nick Bassiliades[3]

[1] Computer Science Department, University of Crete, Heraklion, Greece
[2] Institute of Computer Scince (ICS), Foundation of Research and Technology, Hellas (FORTH), Heraklion, Greece
[3] Department of Informatics, Aristotle University of Thessaloniki, Thessaloniki, Greece

Abstract. The development of the Semantic Web proceeds in steps, building each layer on top of the other. Currently, the focus of research efforts is concentrated on logic and proofs, both of which are essential, since they will allow systems to infer new knowledge by applying principles on the existing data and explain their actions. Research is shifting towards the study of non-monotonic systems that are capable of handling conflicts among rules and reasoning with partial information. As for the proof layer of the Semantic Web, it can play a vital role in increasing the reliability of Semantic Web systems, since it will be possible to provide explanations and/or justifications of the derived answers. This paper reports on the implementation of a system for visualizing proof explanations on the Semantic Web. The proposed system applies defeasible logic, a member of the non-monotonic logics family, as the underlying inference system. The proof representation schema is based on a graph-based methodology for visualizing defeasible logic rule bases.

1 Introduction

The development of the Semantic Web proceeds in steps, building each layer on top of the other. At this point, the highest layer of the Semantic Web is the ontology layer, while research starts focusing on the development of the next layers, the logic and proof layers. The implementation of these two layers is very critical, since they will allow the systems to infer new knowledge by applying principles on the existing data, explaining their actions, sources and beliefs.

Recent trends of research focus mainly on the integration of rules and ontologies, which is achieved with Description Logic Programs (DLPs) [1], [2], [3] or with rule languages like TRIPLE [4] and SWRL [5]. Another interesting research effort involves the standardization of rules for the Semantic Web, which includes the RuleML Markup Initiative [6] and the Rule Interchange Format (RIF) W3C Working Group.

Recently research has been shifted towards the study of non-monotonic systems capable of handling conflicts among rules and reasoning with partial information. Some recently developed non-monotonic rule systems for the Semantic Web are:

N. Bassiliades, G. Governatori, and A. Paschke (Eds.): RuleML 2008, LNCS 5321, pp. 197–210, 2008.

1. DR-Prolog [7] is a system that implements the entire framework of Defeasible Logic, and is thus able to reason with: monotonic and nonmonotonic rules, preferences among rules, RDF data and RDFS ontologies. It is syntactically compatible with RuleML, and is implemented by transforming information into Prolog.
2. DR-DEVICE [8] is also a defeasible reasoning system for the Semantic Web. It is implemented in CLIPS, and integrates well with RuleML and RDF.
3. SweetJess [9] implements defeasible reasoning through the use of situated courteous logic programs. It is implemented in Jess, and allows for procedural attachments, a feature not supported by any of the aforementioned implementations.
4. dlvhex [10] is based on dl-programs, which realize a transparent integration of rules and ontologies using answer-set semantics.

As for the proof layer of the Semantic Web, it has not yet received enough attention, although it can play a vital role in the eventual acceptance of the Semantic Web on behalf of the end-users. More specifically, for a Semantic Web system to be reliable, explanations and/or justifications of the derived answers must be provided. Since the answer is the result of a reasoning process, the justification can be given as a derivation of the conclusion with the sources of information for the various steps. On the other hand, given a reasoning system is able to provide solid proof explanations, it is important to choose an effective and fully expressive representation of the proof to facilitate agent communication.

In this work we describe a system for visualizing proof explanations on the Semantic Web. The proposed system is based on the implementation presented in [11], [12], which uses defeasible logic [13], a member of the non-monotonic logics family, as the underlying inference system. The proof representation schema adopted by our approach is based on [14], a graph-based methodology for visualizing defeasible logic rule bases.

The rest of the paper is organized as follows: Section 2 presents the basic notions of defeasible logics, focusing on its proof theory. The next section discusses the approach followed for generating and representing visualizations of proofs in defeasible logic, accompanied by an example that better illustrates our methodology. The paper ends with a final section that features concluding remarks and poses directions for future research and improvements.

2 Defeasible Logics

2.1 Basics

Defeasible logics is a simple rule-based approach to reasoning with incomplete and inconsistent information. It is suitable to model situations where there exist rules and exceptions by allowing conflicting rules. A superiority relation is used to resolve contradictions among rules and preserve consistency. Formally, a defeasible theory is a triple $(F, R, >)$ where F is a set of literals, R is a finite set of rules and $>$ is a superiority relation on R. There are three kinds of rules:

- Strict rules denoted $A \rightarrow p$ represent rules in the deductive sense. That is, if the premises of the rule are indisputable, the supported literal holds indisputably as well.
- Defeasible rules denoted $A \Rightarrow p$ represent rules that can be defeated by contradicting evidence. That is, when the premises of the rule hold, the conclusion of the rule holds as well unless there exist stronger conflicting evidence.
- Defeaters denoted $A \rightsquigarrow p$ and are used to defeat some defeasible rules by supporting conflicting evidence.

A superiority relation is an acyclic relation $>$ on R that imposes a partial ordering among elements in R. Given two rules r_1 and r_2, if $r_1 > r_2$, we say that r_1 is superior to r_2 and r_2 is inferior to r_1.

2.2 Proof Theory

A conclusion in D is a tagged literal and may have one of the following form:

- $+\Delta q$, meaning q is definitely provable in D.
- $+\partial q$, meaning q is defeasibly provable in D.
- $-\Delta q$, meaning q has proved to be not definitely provable in D.
- $-\partial q$, meaning q has proved to be not defeasibly provable in D.

In order to prove that a literal is definitely provable, we need to establish a proof for q in the classical sense, that is, a proof consisting of facts and strict rules only, and no other matters need to be taken into consideration.

Whenever a literal is definitely provable, it is also defeasibly provable. In that case the defeasible proof for q coincides with the definite proof. Otherwise, in order to prove q defeasibly in D we must find a strict or defeasible rule supporting q that can be applied. In addition, we must also make sure that the specified proof is not overridden by contradicting evidence. Therefore, we must first make sure that the negation of q is not definitely provable in D. Sequentially, we must consider every rule that is not known to be inapplicable and has head $\sim q$. For each such rule s we require that there is a counterattacking rule t with head q with the following properties:

- t must be applicable at this point.
- t must be stronger than s.

To prove that q is not definitely provable in D, q must not be fact, and every strict rule supporting q must be known to be inapplicable.

If a literal q is proved to be not definitely provable, it is also proved to be not defeasibly provable. Otherwise , in order to prove that a literal is not defeasibly provable we first make sure it is not definitely provable. In addition, one of the following conditions must hold:

- None of the rules with head q can be applied
- $\sim q$ is definitely provable

– There is an applicable rule r with head $\sim q$ such that no possibly applicable rule s with head q is superior to r.

A system attempting to provide a graphical representation of a proof explanation based on defeasible reasoning must incorporate all of the aforementioned cases. The challenge of visualizing such a proof explanation lies in the non-monotonicity of the theory that increases the complexity of a well-established proof explanation in comparison to classic, deductive logics. The system developed is described in more detail in the following section.

3 Method

3.1 Tree–Based Proof Explanation in XML

In order to perform reasoning over a defeasible theory and to provide visualization of respective proofs we used a system proposed in [11], [12]. This approach is based on a translation of a defeasible theory into a logic metaprogram as is defined in [15], [16], that works in conjunction with the logic programming system XSB to support defeasible reasoning. When queried upon a literal, XSB produces a trace of all the successful and unsuccessful paths of the proof explanation. The trace tree is pruned to keep only the necessary information of every proof tree, and the pruned proof explanation is then expressed in XML, a meta-language widely used in the Semantic Web.

In their XML schema, they used a similar syntax to RuleML to represent *Facts* and *Rules*. *Atom* element which refers to an atomic formula is used consisting of two elements, an operator element (Op) and a finite set of Variable (Var) or/and Individual constant elements (Ind), preceded optionally by a not statement (in case representation of a negative literal is required). A *Fact* consists of an Atom that comprises certain knowledge. The last primitive entity of the schema is Rule. In defeasible logic, distinction between two kinds of Rules is provided: *Strict Rules* and *Defeasible Rules*. In the proposed schema every kind of rule is noted with a different element. Both kind of rules consist of two parts, the Head element which constitutes of an Atom element, and the Body element which constitutes of a number of Atom elements.

```
<xsd:element name = "Atom">
   <xsd:complexType>
     <xsd:choice>
       <xsd:sequence>
         <xsd:element name= "Op"/>
         <xsd:sequence minOccurs = "0" maxOccurs = "unbounded">
           <xsd:element name= "Var" minOccurs = "0"/>
           <xsd:element name ="Ind" minOccurs = "0"/>
         </xsd:sequence>
       </xsd:sequence>
       <xsd:sequence>
         <xsd:element name="Not">
           <xsd:complexType>
             <xsd:sequence>
               <xsd:element name= "Op"/>
               <xsd:sequence minOccurs = "0" maxOccurs = "unbounded">
                 <xsd:element name= "Var" minOccurs = "0"/>
```

```
                <xsd:element name ="Ind" minOccurs = "0"/>
              </xsd:sequence>
            </xsd:sequence>
          </xsd:complexType>
        </xsd:element>
      </xsd:sequence>
    </xsd:choice>
  </xsd:complexType>
 </xsd:element>

<xsd:element name = "Strict_rule">
   <xsd:complexType>
     <xsd:sequence>
       <xsd:element ref= "Head"/>
       <xsd:element ref= "Body"/>
     </xsd:sequence>
     <xsd:attribute name = "Label" type = "xsd:string" use="required"/>
   </xsd:complexType>
 </xsd:element>

 <xsd:element name = "Defeasible_rule">
   <xsd:complexType>
     <xsd:sequence minOccurs="0">
       <xsd:element ref= "Head"/>
       <xsd:element ref= "Body"/>
     </xsd:sequence>
     <xsd:attribute name = "Label" type ="xsd:string" use="required"/>
   </xsd:complexType>
 </xsd:element>

 <xsd:element name= "Head">
   <xsd:complexType>
     <xsd:sequence>
       <xsd:element ref= "Atom"/>
     </xsd:sequence>
   </xsd:complexType>
 </xsd:element>

 <xsd:element name= "Body">
   <xsd:complexType>
     <xsd:sequence>
       <xsd:element ref= "Atom" minOccurs="0" maxOccurs="unbounded"/>
     </xsd:sequence>
   </xsd:complexType>
 </xsd:element>

 <xsd:element name= "Fact">
   <xsd:complexType>
     <xsd:sequence>
       <xsd:element ref= "Atom"/>
     </xsd:sequence>
   </xsd:complexType>
 </xsd:element>
```

Different elements exists also for each type of proof. More specifically, the *Definitely provable* element consists of the *Atom* to be proven and its *Definite proof*, while the *Definite proof* itself consists either of a Strict rule supporting the Atom to be proven with the respective definite proof for each literal in the rule's body. In case the literal in question is a fact the *Definite proof* consists solely of the corresponding *Fact* element. The *Not Definitely provable* element consists of the *Atom* in question and its *Not Definite proof*. The *Not Definite*

proof consists of all possible strict rules that support the literal in question and the reason they are blocked (*Blocked* element).

```
<xsd:element name = "Definitely_provable" >
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element ref = "Atom" />
        <xsd:element ref = "Definite_Proof" />
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>

  <xsd:element name = "Definite_Proof">
    <xsd:complexType>
      <xsd:choice>
        <xsd:sequence>
          <xsd:element ref= "Strict_rule"/>
          <xsd:element ref= "Definitely_provable" minOccurs="0" maxOccurs="unbounded"/>
        </xsd:sequence>
        <xsd:element ref= "Fact"/>
      </xsd:choice>
    </xsd:complexType>
  </xsd:element>

 <xsd:element name = "Not_Definitely_provable">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element ref = "Atom" />
        <xsd:element ref = "Not_Definite_Proof" />
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>

  <xsd:element name = "Not_Definite_Proof">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element ref= "Blocked" minOccurs="0" maxOccurs="unbounded"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>

  <xsd:element name = "Blocked">
    <xsd:complexType>
      <xsd:choice>
        <xsd:sequence>
          <xsd:element ref="Defeasible_rule"/>
            <xsd:choice>
              <xsd:element ref="Superior"/>
              <xsd:element ref="Not_Defeasibly_provable" />
            </xsd:choice>
        </xsd:sequence>
        <xsd:sequence>
          <xsd:element ref="Strict_rule"/>
          <xsd:element ref= "Not_Definitely_provable"/>
        </xsd:sequence>
        <xsd:element name="Not_Superior">
          <xsd:complexType>
            <xsd:sequence>
              <xsd:element ref="Defeasible_rule"/>
            </xsd:sequence>
          </xsd:complexType>
        </xsd:element>
      </xsd:choice>
    </xsd:complexType>
  </xsd:element>
```

A *Defeasibly provable* element consists of the *Atom* to be proven and its *Defeasible proof*. The *Defeasible proof* consists of the applicable rule supporting the *Atom* to be proven and its *Defeasible proof*, followed by a *Not Definitely provable* element concerning the negation of *Atom*, and a sequence of *Blocked* elements for every rule that is not known to be inapplicable and has head the negation of the *Atom* in question.

```
<xsd:element name = "Defeasibly_provable" >
   <xsd:complexType>
     <xsd:choice>
       <xsd:element ref="Definitely_provable"/>
       <xsd:sequence>
         <xsd:element ref = "Atom" />
         <xsd:element ref = "Defeasible_Proof" />
       </xsd:sequence>
     </xsd:choice>
   </xsd:complexType>
 </xsd:element>

 <xsd:element name = "Defeasible_Proof">
   <xsd:complexType>
     <xsd:sequence>
       <xsd:choice>
         <xsd:element ref= "Strict_rule"/>
         <xsd:element ref= "Defeasible_rule"/>
       </xsd:choice>
       <xsd:element ref="Defeasibly_provable" minOccurs="0" maxOccurs="unbounded"/>
       <xsd:element ref="Not_Definitely_provable"/>
       <xsd:element ref="Blocked" minOccurs="0" maxOccurs="unbounded"/>
     </xsd:sequence>
   </xsd:complexType>
 </xsd:element>
```

A *Not Defeasibly provable* element consists of the *Atom* in question and its *Not Defeasible proof*. The *Not Defeasible proof* consists of the *Not Definitely provable* element for the *Atom* in question and either a sequence of *Blocked* elements for every rule with head the *Atom* in question and the reason they cannot be applied, or a *Definitely provable* element for the negation of the *Atom*, or by the element *Undefeated* providing an applicable rule r with head the negation of *Atom* in question such that no possibly applicable rule s with head the specified *Atom* is superior to r.

```
 <xsd:element name = "Superior">
   <xsd:complexType>
     <xsd:sequence>
       <xsd:element ref="Defeasible_rule"/>
       <xsd:element ref="Defeasibly_provable" minOccurs="0" maxOccurs="unbounded"/>
     </xsd:sequence>
   </xsd:complexType>
 </xsd:element>

 <xsd:element name = "Not_Defeasibly_provable">
   <xsd:complexType>
     <xsd:sequence>
       <xsd:element ref = "Atom" />
       <xsd:element ref = "Not_Defeasible_Proof" />
     </xsd:sequence>
   </xsd:complexType>
 </xsd:element>

 <xsd:element name = "Not_Defeasible_Proof">
```

```
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element ref= "Not_Definitely_provable"/>
        <xsd:choice>
          <xsd:element ref= "Blocked" minOccurs="0" maxOccurs="unbounded"/>
          <xsd:element ref="Definitely_provable"/>
          <xsd:element ref="Undefeated"/>
        </xsd:choice>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>

  <xsd:element name="Undefeated">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element ref= "Defeasible_rule"/>
        <xsd:element ref="Defeasibly_provable" minOccurs="0" maxOccurs="unbounded"/>
        <xsd:element ref= "Blocked" minOccurs="0" maxOccurs="unbounded"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
```

3.2 XML Proof Processing

Visualizing a proof explanation requires information about its structure, which must be extracted from the XML document, produced by the system [11], [12] based on the aforementioned XSD Schema. Due to the recursiveness of the xsd, an Recursive Descent Parser (RDP) was implemented. An RDP is a top-down parser built from a set of mutually-recursive procedures (or a non-recursive equivalent) where each such procedure usually implements one of the production rules of the grammar. Thus the structure of the resulting program closely mirrors that of the grammar it recognizes. In our system, the RDP parses the XML document with the assistance of Xerces, the Open Source Apache project's XML parser, and stores the main proof as well as each secondary proof in a different tree–shape structure. Each such structure holds the information required to represent the corresponding proof explanation, i.e. the sequence of rules participating in the proof. For each rule the following information is held:

- the name
- the type (definite or defeasible)
- the head
- the body
- the names of the attacking rules that could defeat it — if such rules exist, or whether the rule is undefeated.

After parsing the XML document and keeping all the appropriate information, the visualization of every proof takes place. In order to visualize every proof, since we consider it has a tree–shape structure, we need to evaluate the height of each node of the proof in the tree. Each node is considered to be either an atom that participates in the body or the head of some rule or the rule itself. For the visualization of the components of the rules , we used the library of [14] which renders each node and the connections between them.

In their approach the digraph representation contains two kinds of nodes:

- literals, represented by rectangles, which they call literal boxes
- rules, represented by circles

Each literal box consists of two adjacent atomic formula boxes, with the upper one of them representing a positive atomic formula and the lower one representing a negated atomic formula.

3.3 Visualization

Definite Proofs. A definite proof is the simplest case of proof explanation. Such a proof consists either of a fact of the literal in question or of a sequence of Strict Rules that fire proving the literal in question.

Not Definite Proofs. A not-definite proof consists of a sequence of proof explanations. Each proof explanation shows why the Strict Rules with head equal to the negation of the literal in question do not fire.

Defeasible Proofs. As mentioned above, a literal is defeasibly provable either if it is already definitely provable or the defeasible part needs to be argued upon. In case the literal is definitely provable, the proof is visualized as described in section 3.3. Otherwise, in order to produce a fully descriptive visualization of a defeasible proof faithful to the reasoning referred to above, several parts are included in the graphic:

1. The main proof, i.e. the sequence of defeasible or definite rules supporting the literal in question.
2. A not definite proof for the negation of the literal in question.
3. A series of not defeasible proofs for every rule attempting to prove the negation of the literal in question. If the specified rule does not fire, the proof consists of a chain of rules supporting a literal in the rule's body that is not provable. Otherwise, if the rule fires but is defeated by a superior applicable counterattacking rule, the proof consists of both the chain of rules proving the negation of the literal and the chain of rules proving the literal in question, and the superiority relation is displayed verbally.

Seperate proofs are visualized in seperate tabs in the graphic.

Not Defeasible Proofs. As described in section 2, a series of conditions must hold in order for a literal to be not defeasibly provable. The visualization of a Non Defeasible Proof consists of two parts:

1. The visualization of the Not Definite Proof for the specified literal.
2. If the literal is not defeasibly provable because all rules that support it do not fire, a Not Defeasible Proof for each such rule is visualized in a seperate tab. If the negation of the literal is definitely provable, then a seperate tab visualizing the Definite Proof is included in the graphic. Otherwise, if there

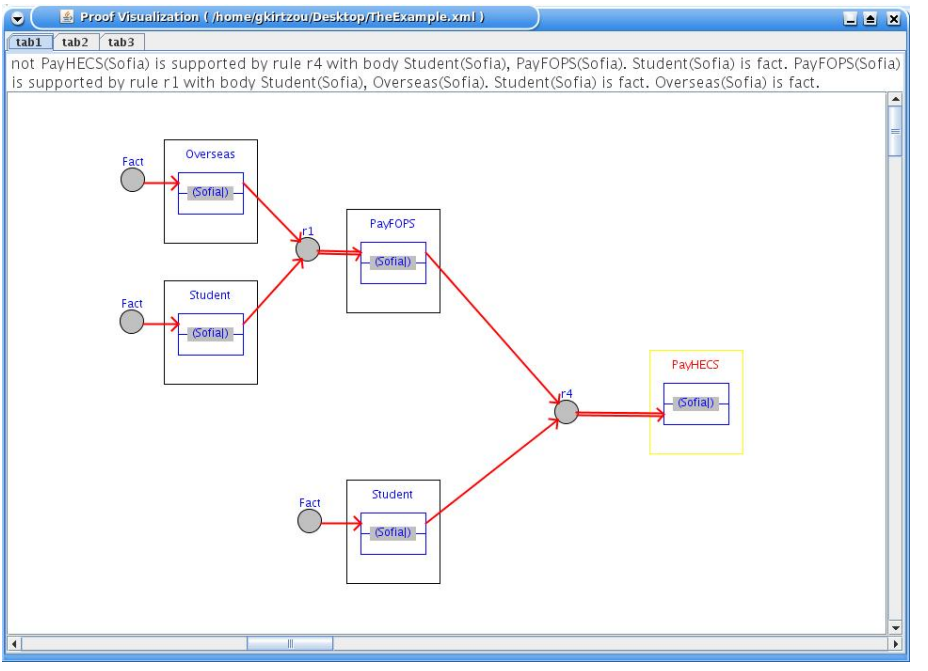

Fig. 1. The main defeasible proof of ∼payHECS(Sofia)

exists an undefeated rule supporting the negation of the literal, the corresponding Defeasible Proof is included in the graphic, and a series of Not Defeasible Proofs for each counterattacking rule with a not defeasibly provable body on separate tabs. Rules that fail to fire because they are defeated by the undefeated rule appear on seperate tabs as well, and the superiority relationship is expressed verbally.

3.4 Example

To demonstrate our tool, we are using as an example a subset of the knowledge base given in [17], modelling part of the Griffith University guidelines on fees. In particular, we consider the following rules:

- r_1: student(X), overseas(X) ⇒payFPOS(X)
- r_2: student(X), overseas(X), exchange(X) ⇒∼ payFPOS(X)
- r_3: student(X) ⇒ payHECS(X)
- r_4: student(X), payFPOS(X) ⇒∼payHECS(X)
- $r_4 > r_3$

The rules represent the following policy:
Overseas students generally pay Overseas Students Fee (FPOS),unless they come from an international exchanged program. All students pay the Higher Education Contribution Scheme (HECS), apart from students who pay FPOS.

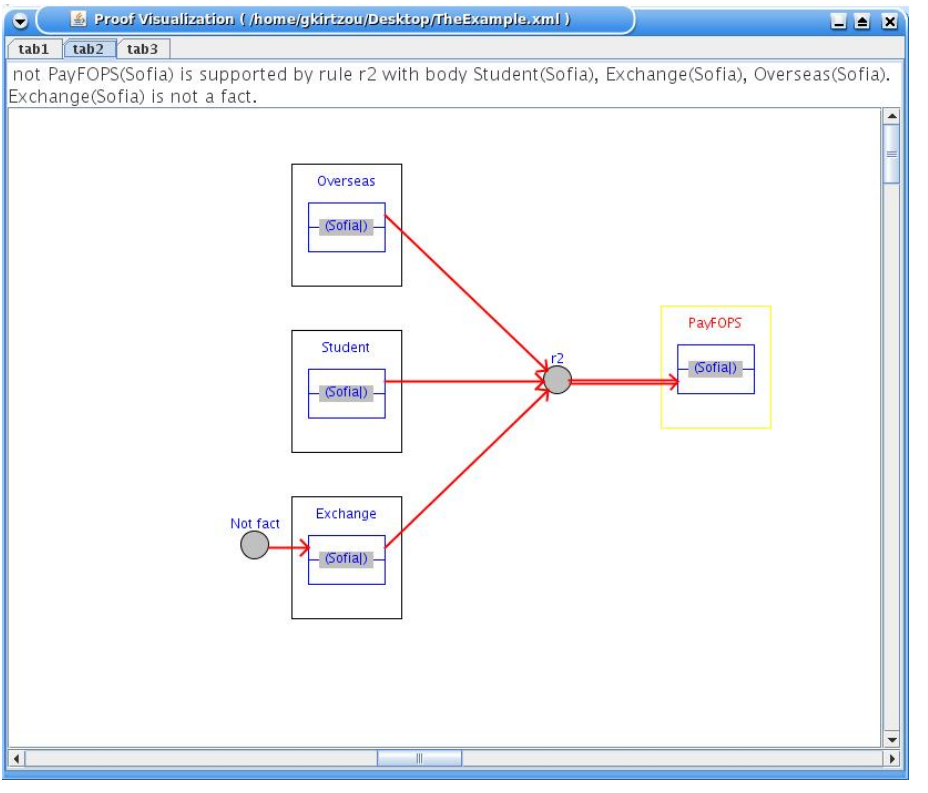

Fig. 2. A not definite proof for the negation of the literal payFPOS(Sofia), due to the recursiveness of defeasible proof

Suppose we have an overseas student named Sofia and we want to query upon whether she has to pay HECS or not. Our knowledge base now comprises of the aforementioned rules in addition to the following facts:

- student(Sofia)
- overseas(Sofia)

∼payHECS(Sofia) is defeasibly provable, since rule r_1 fires establishing the literal payFPOS(Sofia). Therefore, rule r_4 fires supporting payHECS(Sofia). The first tab of the GUI (Figure1) illustrates this sequence of applicable rules. The tree–shaped structure of the proof is achieved with the duplication of the required nodes(literals), leading to a more easy reading form of visualization. Rule r_2 supporting ∼payFPOS(Sofia)) does not fire, since Sofia is not an exchange student in our knowledge base. This is demonstrated in the second tab of of our GUI (Figure 2). Rule r_3 supportingpayHECS(Sofia) fires, but loses due to superiority relation. This renders on the third tab of the GUI (Figure 3). The seperate components of this proof are also presented verbally at the top of each tab. As we mentioned in section 3.3 the superiority relationships are presented only verbally.

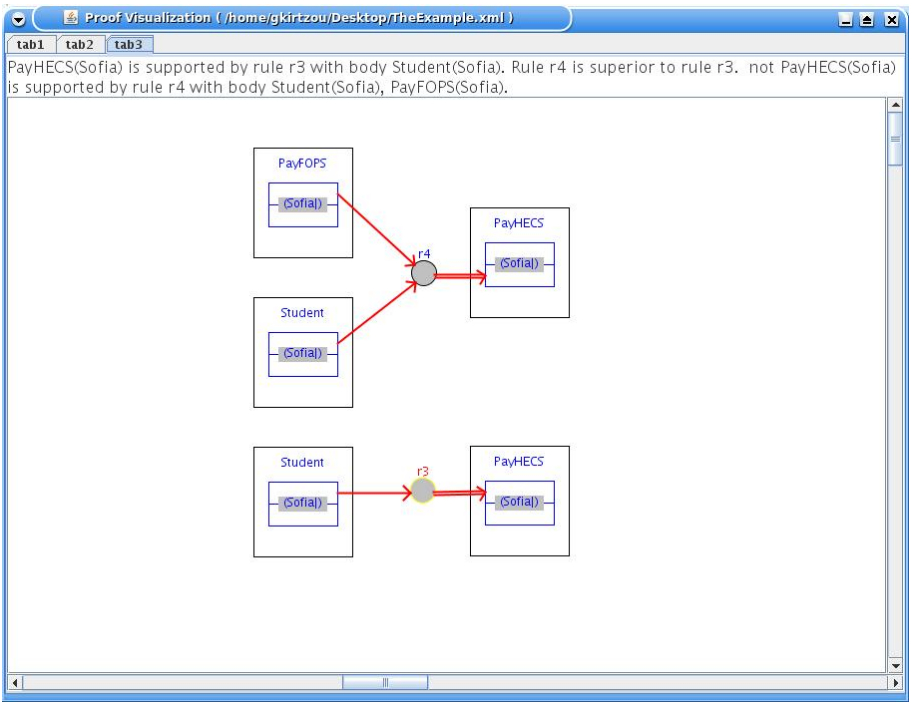

Fig. 3. The not defeasible proof for the rule r_3, which is attempting to prove the negation of the literal in question, payHECS(Sofia)

4 Conclusions and Future Work

This paper attempts to fill the apparent gap in the development of the proof layer of the Semantic Web, by presenting a system for visualizing proofs in the Semantic Web environment. The proposed system uses defeasible logic as the underlying inference system and its adopted proof representation schema is based on enhanced directed graphs that feature a variety of node and connection types for expressing the necessary elements of the defeasible logic proof theory. More specifically, the system offers the capability of visualizing both definite (facts and strict rules only) and non-definite proofs (sequence of proof explanations that show why strict rules with head equal to the negation of the literal in question do not fire) as well as defeasible (either definitely provable or the defeasible part needs to be argued upon) and non-defeasible proofs (not definitely provable plus some additional conditions). Section 2 provides a deeper insight on defeasible logic proof theory. An example was also provided that demonstrates the representational capabilities of the proposed implementation.

As for future directions, there is still room for improvement. The visual representation should incorporate some further elements of defeasible reasoning,

like the superiority relationship, which is currently only displayed verbally. Additionally, the proposed software tool should undergo a thorough user evaluation, in order to assess the degree of expressiveness it offers and whether the derived proof visualizations are indeed more comprehensible than the XML-based proofs. An interesting idea would also involve the integration into the system of a visual defeasible theory representation tool, like the one presented in [18]. Then, users would have the ability of (visually) querying the system regarding the proof status of every literal in the rule base and observe a visualization of the corresponding proof trace.

References

1. Grosof, B.N., Horrocks, I., Volz, R., Decker, S.: Description Logic Programs: Combining Logic Programs with Description Logic. In: WWW 2003: Proceedings of the 12th international conference on World Wide Web, pp. 48–57. ACM, New York (2003)
2. Levy, A.Y., Rousset, M.C.: Combining Horn rules and description logics in CARIN. Artificial Intelligence 104(1-2), 165–209 (1998)
3. Rosati, R.: On the decidability and complexity of integrating ontologies and rules. J. Web Sem. 3(1), 61–73 (2005)
4. Sintek, M., Decker, S.: TRIPLE - A Query, Inference, and Transformation Language for the Semantic Web. In: Horrocks, I., Hendler, J. (eds.) ISWC 2002. LNCS, vol. 2342, pp. 364–378. Springer, Heidelberg (2002)
5. Horrocks, I., Patel-Schneider, P.F.: A Proposal for an OWL Rules Language. In: WWW 2004: Proceedings of the 13th international conference on World Wide Web, pp. 723–731. ACM, New York (2004)
6. RuleML: The RuleML Initiative website (2006), `http://www.ruleml.org/`
7. Antoniou, G., Bikakis, A.: DR-Prolog: A System for Defeasible Reasoning with Rules and Ontologies on the Semantic Web. IEEE Trans. on Knowl. and Data Eng. 19(2), 233–245 (2007)
8. Bassiliades, N., Antoniou, G., Vlahavas, I.: A Defeasible Logic Reasoner for the Semantic Web. International Journal of Semantic Web and Information Systems (IJSWIS) 2(1), 1–41 (2006)
9. Gandhe, M., Finin, T., Grosof, B.: SweetJess: Translating DamlRuleML to Jess. In: International Workshop on Rule Markup Languages for Business Rules on the Semantic Web in conjunction with ISWC 2002, Sardinia, Italy (2002)
10. Eiter, T., Ianni, G., Schindlauer, R., Tompits, H.: Dlvhex: A System for Integrating Multiple Semantics in an Answer-Set Programming Framework. In: Fink, M., Tompits, H., Woltran, S. (eds.) WLP, Technische Universität Wien, Austria. INFSYS Research Report, vol. 1843, pp. 206–210 (2006)
11. Antoniou, G., Bikakis, A., Dimaresis, N., Genetzakis, M., Georgalis, G., Governatori, G., Karouzaki, E., Kazepis, N., Kosmadakis, D., Kritsotakis, M., Lilis, G., Papadogiannakis, A., Pediaditis, P., Terzakis, C., Theodosaki, R., Zeginis, D.: Proof Explanation for the Semantic Web Using Defeasible Logic. In: Zhang, Z., Siekmann, J.H. (eds.) KSEM 2007. LNCS (LNAI), vol. 4798, pp. 186–197. Springer, Heidelberg (2007)

12. Antoniou, G., Bikakis, A., Dimaresis, N., Genetzakis, M., Georgalis, G., Governatori, G., Karouzaki, E., Kazepis, N., Kosmadakis, D., Kritsotakis, M., Lilis, G., Papadogiannakis, A., Pediaditis, P., Terzakis, C., Theodosaki, R., Zeginis, D.: Proof explanation for a nonmonotonic Semantic Web rules language. Data Knowl. Eng. 64(3), 662–687 (2008)
13. Nute, D.: Defeasible logic. In: Gabbay, D., Hogger, C.J., Robinson, J.A. (eds.) Handbook of Logic in Artificial Intelligence and Logic Programming. Nonmonotonic Reasoning and Uncertain Reasoning, vol. 3, pp. 353–395. Oxford University Press, Oxford (1994)
14. Kontopoulos, E., Bassiliades, N., Antoniou, G.: Visualizing Defeasible Logic Rules for the Semantic Web. In: Mizoguchi, R., Shi, Z., Giunchiglia, F. (eds.) ASWC 2006. LNCS, vol. 4185, pp. 278–292. Springer, Heidelberg (2006)
15. Antoniou, G., Billington, D., Governatori, G., Maher, M.J.: Embedding Defeasible Logic into Logic Programming. Theory Pract. Log. Program. 6(6), 703–735 (2006)
16. Maher, M.J., Rock, A., Antoniou, G., Billington, D., Miller, T.: Efficient Defeasible Reasoning Systems. International Journal on Artificial Intelligence Tools 10(4), 483–501 (2001)
17. Antoniou, G., Billington, D., Maher, M.J.: On the Analysis of Regulations using Defeasible Rules. In: HICSS (1999)
18. Bassiliades, N., Kontopoulos, E., Antoniou, G.: A Visual Environment for Developing Defeasible Rule Bases for the Semantic Web. In: Adi, A., Stoutenburg, S., Tabet, S. (eds.) RuleML 2005. LNCS, vol. 3791, pp. 172–186. Springer, Heidelberg (2005)