# A Visualization Algorithm for Defeasible Logic Rule Bases over RDF Data

Efstratios Kontopoulos[1], Nick Bassiliades[1], Grigoris Antoniou[2]

[1]Department of Informatics
Aristotle University of Thessaloniki
GR-54124 Thessaloniki, Greece
{skontopo,nbassili}@csd.auth.gr

[2]Institute of Computer Science, FO.R.T.H.
P.O. Box 1385, GR-71110,
Heraklion, Greece
antoniou@ics.forth.gr

**Abstract.** This work presents a visualization algorithm for defeasible logic rule bases as well as a software tool that applies this algorithm, according to which, a directed graph is produced that represents the rule base. The graph features distinct node types for rules and atomic formulas and distinct connection types for the various rule types of defeasible logic.

## 1. Introduction

Logic and proofs posses a key role in the acceptance of the Semantic Web on behalf of the users. *Defeasible reasoning* [3] represents a rule-based approach to reasoning with incomplete, changing and conflicting information. Nevertheless, it is based on solid mathematical formulations and is, thus, not fully comprehensible by users, who often need graphical trace and explanation mechanisms for the derived conclusions.

This paper presents a visualization algorithm for defeasible logic rule bases and a software tool that applies this algorithm. For the representation of the rule base, *directed graphs* are applied that feature distinct node and connection types. The tool is called *dl-RuleViz* and is implemented as part of *VDR-DEVICE* [1], an environment for modeling and deploying defeasible logic rule bases on top of RDF ontologies.

## 2. Visualizing a Defeasible Logic Rule Base

The full theoretical approach, regarding the graphical representation of defeasible reasoning elements was discussed in a previous work of ours [2]. For every class in the rule base, a *class box* with the same name is constructed. Class boxes are containers, which are dynamically populated with one or more *class patterns*. Class patterns express conditions on *filtered subsets* of instances of the specific class and are populated with one or more *slot patterns*. Slot patterns represent conditions on slots (or class properties) and they consist of a slot name and, optionally, a variable and a list of value constraints. The variable is used for unifying the slot value, with the latter having to satisfy the list of constraints.

For the placement of each element in the graph, an algorithm (Fig. 1) for the visualization of the defeasible logic rule base is proposed that takes advantage of common rule stratification techniques. Unlike the latter, however, that focus on computing the minimal model of a rule set, our algorithm aims at the optimal *visualization* outcome.

```
str:=1
foreach cb∈$CB_b$ do stratum(cb):=str
while |RS|≠0 do
  RuleTemp:=∅
  str:=str+1
  foreach R∈RS do
    if   ((∀p∈premises(R) → stratum(class(p))<str) ∧
         (∃p'∈premises(R) ∧ stratum(class(p'))=str-1))
    then stratum(R):=str, RS:=RS-{R}, RuleTemp:=RuleTemp ∪{R}
  foreach R∈RuleTemp do
    foreach p∈premises(R) do
         if stratum(class(p))=str-1 then Type:=plain else Type:=expandable,
         in-arrow(R):=in-arrow(R)∪{<p,Type>},
         out-arrow(p):=out-arrow(p)∪{<R,Type>},
  str:=str+1
  CbTemp:=∅
  foreach R∈RuleTemp do
     if unknown(stratum(class(conclusion(R))))
     then stratum(class(conclusion(R))):=str, CbTemp:=CbTemp∪{class(conclusion(R))}
  foreach R∈RuleTemp do
    if type(R)=strictrule then Type:= strict
    else if type(R)=defeasible then Type:=defeasible
    else Type:=defeater,
    if class(conclusion(R))∈CbTemp then Orient:=plain else Orient:=dotted,
    out-arrow(R):=out-arrow(R)∪{<conclusion(R),Orient,Type>},
    in-arrow(class(conclusion(R))):=in-arrow(conclusion(R))∪{<R,Orient,Type>}
```

**Fig. 1.** The rule stratification algorithm

The algorithm gives a left-to-right orientation to the flow of information in the graph by "*stratifying*" the graph elements, i.e. by calculating the optimal stratum, where each graph element has to be placed. The following steps can be distinguished:

1. All base class boxes are placed in stratum #1.
2. The algorithm enters a loop, consecutively assigning strata to rule circles and derived class boxes, incrementing each time the stratum counter by 1.
   a. A rule circle is assigned to a stratum, if all its premises belong to previous strata, with at least one of them belonging to the immediately previous stratum.
   b. A class box is assigned to a stratum, if it contains the conclusions of rules in the immediately previous stratum.

When a conclusion of a rule serves as a premise for another rule in a previous stratum, the conclusion is not drawn again and the arrow connecting the rule with the conclusion is not drawn backwards. Instead, a "*dotted*" arrow is drawn, commencing from the rule circle and ending in three dots "**…**", to reduce complexity. By clicking on the arrow, a pop-up window shows the rule isolated in its completeness.

Only the arcs that connect two *consecutive* graph elements are drawn by default. When the stratum difference between a class pattern and a rule circle is greater than 1, the arrow that connects them is "*expandable*". To prevent graph cluttering, expandable arrows are drawn only at the user's request.

For example, suppose that we have the following rule base:

$r_1$: novel(X) $\rightarrow$ book(X)

$r_2$: book(X) $\Rightarrow$ hardcover(X)        $r_3$: novel(X) $\Rightarrow$ $\neg$hardcover(X)

```
r₄: novel(X),collectible(X,“yes”) ⇒ rare(X)
r₅: novel(X),author(X,“Asimov”),price(X,Y),Y>18 ⇒ hardcover(X)
```

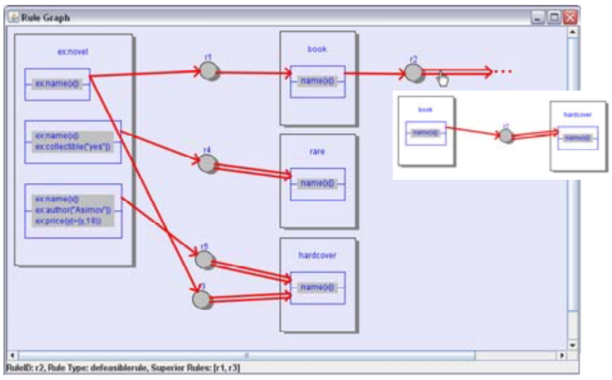

**Fig. 2.** Implementation of the visualization algorithm by dl-RuleViz

After applying the algorithm, it comes up that four strata (or columns) are needed to display all the graph elements. The resulting graph (Fig. 2), produced by dl-RuleViz, is compliant with the algorithm presented. The pop-up window displays the premises and conclusion of rule $r_2$.

## 3. Future Work

Potential improvements of dl-RuleViz and the visualization algorithm include enhancing the derived graph with negation-as-failure and variable unification, for simplifying the display of multiple unifiable class patterns. Expressive visualization of a defeasible logic rule base can then lead to proof explanations. By adding visual rule execution tracing, proof visualization and validation to the dl-RuleViz module, we can delve deeper into the Proof layer of the Semantic Web architecture, implementing facilities that would increase the trust of users towards the Semantic Web.

## References

[1] Bassiliades N., Kontopoulos E., Antoniou G., “A Visual Environment for Developing Defeasible Rule Bases for the Semantic Web”, *Proc. RuleML-2005*, Galway, Ireland, Springer-Verlag, LNCS 3791, pp. 172-186, 2005.

[2] Kontopoulos E., Bassiliades N., Antoniou G., “Visualizing Defeasible Logic Rules for the Semantic Web”, *1st Asian Semantic Web Conference (ASWC'06)*, Beijing, China, Springer-Verlag, LNCS 4185, pp. 278-292, 2006.

[3] Nute D., “Defeasible Reasoning”. *Proc. 20th Int. Conference on Systems Science*, pp. 470-477, IEEE Press, 1987.