Knowledge-Based Systems 24 (2011) 406–419

Contents lists available at ScienceDirect

Knowledge-Based Systems

journal homepage: www.elsevier.com/locate/knosys

Visualizing Semantic Web proofs of defeasible logic in the DR-DEVICE system

Efstratios Kontopoulos [a,*], Nick Bassiliades [a], Grigoris Antoniou [b]

[a] Department of Informatics, Aristotle University of Thessaloniki, GR-54124 Thessaloniki, Greece
[b] Institute of Computer Science, FO.R.T.H., P.O. Box 1385, GR-71110 Heraklion, Greece

ARTICLE INFO

Article history:
Received 17 May 2010
Received in revised form 14 November 2010
Accepted 3 December 2010
Available online 9 December 2010

Keywords:
Semantic Web
Defeasible logic
Explanations
Proof visualization
Rule base

ABSTRACT

The Semantic Web aims at improving the current Web, by augmenting its content with semantics and encouraging the cooperation among human users and machines. Since the basic Semantic Web infrastructure is reaching sufficient maturity, research efforts are shifting towards logic, proof and trust and rule-based systems inevitably concentrate most of the attention. Nevertheless, in order for human users to trust system answers, they have to be presented with adequate explanations that justify the derived results. And, even more importantly, these explanations have to be presented in a user-comprehensible format. Consequently, the focus in this work is on humans and the research area called proof visualization that features three main approaches: tree-based, graphical and logical/textual. Since each of the approaches presents advantages and disadvantages, this article proposes a fourth, hybrid visualization approach that combines the pros of all three approaches and attempts to leverage the respective cons. The article also presents a software tool that implements the proposed hybrid approach. The tool is called VProof$_H$ and visualizes defeasible logic proofs, offering multiple representations that adapt to user needs. Extensive scalability and user evaluation tests prove the software tool's usability.

1. Introduction

The *Semantic Web* (*SW*) [10] represents an initiative to improve the current Web, by augmenting web content with semantics, thus, encouraging cooperation among human and software agents. Its basic infrastructure (content representation, ontologies) has acquired sufficient maturity and research efforts are now shifting towards logic, proof and trust, as is demonstrated by various pieces of work (e.g. [35,45,46,22]). Consequently, rule-based systems are gradually gaining popularity, but this development raises the question: can users indeed trust system answers?

While in some cases a system answer is evident and easily justifiable, in most other cases users cannot be confident in the answer, unless they can understand the reasons that led the system to produce such a result. Thus, in order for an answering system to gain the trust of a user: (a) it must be able to provide *explanations* for the generated results, and (b) the explanations have to be presented efficiently in a user-comprehensible format [49,25].

Traditionally, research on explanations was tightly coupled with the development of knowledge-based and expert systems [51,32], although the term "explanation" has also been associated with various other disciplines, like cognitive sciences, linguistics and teaching [44].

With the recent emergence of the Semantic Web, the need for explanation facilities is now more vital than ever. Answers produced from SW systems are typically the result of a reasoning process. Therefore the justification can be given as a derivation of the conclusion with the sources of information for each inference step. The series of explanations forms the *inference proof* and is generated by the reasoning engine that has the capability of taking any particular answer and tracing back through the inference steps used, looking at their antecedents and determining all of the sources used to arrive at an answer.

According to Berners-Lee et al. [11], there are two types of proofs: *machine-processible* (*mechanical*) and *human-oriented* (*proof-as-picture*). In order for an inference process/service to be considered trustworthy, the mechanical proof type should be generated together with the corresponding system results. It should also be provided in a sharable, portable and, preferably, distributed format, which will allow its use by various, heterogeneous applications. On the other hand, a human-oriented (a.k.a. visual) proof presentation (proof-as-picture) would be more suitable for human consumption, so that the user can inspect the proof and retrace and verify the derivation of answers himself. This research area is called proof (trace) visualization and is specifically addressed to human users and not to machines/agents, for which processing proof files is not as demanding as to their human counterparts.

* Corresponding author. Tel.: +30 2310998433; fax: +30 2310998419.
E-mail addresses: skontopo@csd.auth.gr (E. Kontopoulos), nbassili@csd.auth.gr (N. Bassiliades), antoniou@ics.forth.gr (G. Antoniou).

0950-7051/$ - see front matter © 2010 Elsevier B.V. All rights reserved.
doi:10.1016/j.knosys.2010.12.001

McGuinness [34] identifies three main user categories that can be benefited by a proof trace:

1. *Human users or agents*, as mentioned above, needing to decide if they can trust the inference processes used to retrieve information.
2. *Application developers* (authors of reasoners, search engines, database systems), who wish to defend the credibility of their systems
3. *Authors of hybrid solutions programs* (ontology builders who are merging or extending ontologies, developers combining databases or knowledge-based systems), who need to verify how answers were derived and might integrate.

Three major proof presentation approaches are encountered in practice:

1. *Tree Proof Representation* (*TPR*), which better matches the concept of proof trees.
2. *Graphical Proof Representation* (*GPR*) that enriches user-friendliness, by utilizing various graphical elements and visual aids.
3. *Logical Proof Representation* (*LPR*), a textual, logic-based representation.

As presented later, these approaches offer certain advantages and disadvantages. The paper proposes a fourth, novel *hybrid* approach for proof visualization that combines benefits from all representations, offering multiple simultaneous execution trace views. Also, a software tool called *VProof$_H$* is presented here that visualizes defeasible logic proofs, offering various representation/visualization approaches that adapt to user needs. Consequently, VProof$_H$ is aimed at all three user categories described above, as well as other types of users who can potentially be benefited by explanation and proof visualizations.

Defeasible logic [36] is a non-monotonic, rule-based approach for reasoning with incomplete and inconsistent information and is considered as a prominent tool for the SW. This is also confirmed by various recently developed non-monotonic systems [18], like *DR-DEVICE* [6], *DR-Prolog* [12], *SweetJess* [23] and *SPINdle* [31]. VProof$_H$ visualizes proofs generated by the DR-DEVICE defeasible logic reasoner, although it can, potentially, be modified for visualizing defeasible logic proofs produced by other defeasible logic inference engines as well.

The tool applies the DRVgraph defeasible logic rule base visualization framework [28,30], but for visually representing defeasible logic proofs instead. Proofs of specific conclusions are quite different from complete rule bases, in the sense that they use parts of the rule base and contain constants in place of variables. The proofs are based on the Proof XML Schema (PXS), which extends the defeasible logic proof schema proposed in [7], while the derived proof graphs are visually stratified, by adapting the visual stratification algorithm for defeasible rule bases presented in [29] to defeasible logic proofs.

The rest of this article consists of the following: Section 2 gives an insight into the basic concepts of defeasible logics and details regarding its proof theory. Section 3 reports on the three dominant proof representation approaches (TPR, GPR and LPR) and offers a list of popular software tools that apply these approaches, followed by a qualitative comparison. Section 4 presents our hybrid proof visualization approach, focusing on its fundamental principles and the VProof$_H$ system that deploys this approach. Sections 5 and 6 describe the scalability testing and user evaluation performed on the tool, while the paper is concluded with the last section that contains the final remarks, as well as directions for future improvements.

2. Defeasible logics

Defeasible logics are mostly suitable in modeling situations, where there exist rules and exceptions that are expressed via conflicting rules. A superiority relation is used to preserve consistency and resolve these contradictions among rules.

2.1. Basic concepts

A defeasible theory D (a knowledge base in defeasible logic, or a defeasible logic program) is formally represented by a triple $(F, R, >)$, where F is a set of literals (facts), R is a set of rules and $>$ is a superiority relation on R. Rules are represented as r: $A(r) \dashrightarrow C(r)$, where r is a unique rule label, $A(r)$ is the rule body (antecedent), which consists of a finite set of literals, an arrow $\dashrightarrow$ that denotes the logical implication operation and is a placeholder for concrete arrows introduced below and the rule head (consequent) $C(r)$, which is a single literal.

There are three kinds of rules in defeasible logic, each represented by a different arrow:

- *Strict rules*, denoted by $A(r) \rightarrow C(r)$, which represent rules in the deductive sense.
- *Defeasible rules*, denoted by $A(r) \Rightarrow C(r)$, which represent rules that can be defeated by stronger contradicting evidence.
- *Defeaters*, denoted by $A(r) \rightsquigarrow C(r)$, which are used in defeating defeasible rules.

The *superiority relation* is an acyclic, binary relation on R that imposes a partial ordering among R elements. More specifically, given two rules r_1 and r_2, if $r_1 > r_2$, then r_1 is considered *superior* to r_2 and r_2 is *inferior* to r_1. The superiority relation is used in resolving conflicts among competing rules (e.g. rules with complementary heads).

Additionally, another important element in defeasible logic is the notion of *conflicting literals*. In various applications, literals are often considered to be conflicting and at most one of a certain set should be derived. For example, in a price negotiation application, where offers are made by potential buyer(s), only one offer should eventually be concluded. Thus, if possible alternative offers are determined by rules, whose conditions may or may not be mutually exclusive, only one of the rules should ultimately prevail. In this case, the conflict set is: $C(offer(x,y)) = \{\neg offer(x,y)\} \cup \{offer(x,z) \mid z \neq y\}$. The conflict set for the literal $offer(x, y)$ contains the negation of the literal along with all the other offers that are different from $offer(x, y)$.

Finally, since defeasible logics deal with potential conflicts and inconsistencies, they feature *classical negation* and they also deal with *negation-as-failure* (*NAF*), which is typical for non-monotonic logic programming systems. Although NAF is often excluded from the object language of defeasible logics, it can be easily simulated when necessary [3].

2.2. Proof theory

A conclusion in D is a tagged literal and may have one of the following forms [2]:

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

In order to prove $+\Delta q$, a proof for q consisting of facts and strict rules needs to be established. Whenever a literal is definitely provable, it is also defeasibly provable. In that case, the defeasible proof coincides with the definite proof for q. Otherwise, in order to prove $+\partial q$ in D, an applicable strict or defeasible rule supporting q must exist. In addition, it should also be ensured that the specified proof is not overridden by contradicting evidence. Therefore, it has to be guaranteed that the negation of q is not definitely provable in D. Successively, every rule that is not known to be inapplicable and has head $\sim q$ has to be considered. For each such rule s, it is required that there is a counterattacking rule t with head q that is applicable at this point and s is inferior to t.

In order to prove $-\Delta q$ in D, q must not be a fact and every strict rule supporting q must be known to be inapplicable. If it is proved that $-\Delta q$, then it is also proved that $-\partial q$. Otherwise, in order to prove that $-\partial q$, it must firstly be ensured that $-\Delta q$. Additionally, one of the following conditions must hold: (i) None of the rules with head q can be applied, (ii) It is proved that $-\Delta\sim q$, and (iii) There is an applicable rule r with head $\sim q$, such that no possibly applicable rule s with head q is superior to r.

2.3. Example

This section presents an example regarding the inference and proof processes of defeasible logics, which will serve as a running example throughout the paper. Assume that a user submits the following defeasible theory D (presented in simple logical notation) to a defeasible logic reasoner and wants to find out why the conclusion $\neg payBonus(john)$ is defeasibly derived.

$\mathtt{f_1}$: employee(john)
$\mathtt{f_2}$: overtime(john)
$\mathtt{r_1}$: employee(X), overtime(X) $\Rightarrow$ payOvertime(X)
$\mathtt{r_2}$: employee(X), overtime(X), newEmployee(X)
 $\Rightarrow$ ¬payOvertime(X)
$\mathtt{r_3}$: employee(X) $\Rightarrow$ payBonus(X)
$\mathtt{r_4}$: employee(X), payOvertime(X) $\Rightarrow$ ¬payBonus(X)
$\mathtt{r_4} > \mathtt{r_3}$

The final conclusion is $+\partial\neg payBonus(john)$, which was derived by rule r_4, whose body literals were also defeasibly proved due to the existence of fact f_1 and a defeasible proof due to rule r_1. Rule r_1 generates $+\partial payOvertime(john)$, since its body literals are defeasibly proved via the existence of facts f_1 and f_2. Furthermore, the opposite conclusion $payBonus(john)$ was not proved, since r_3 was attacked by superior rule r_4. Rule r_2 is blocked, because one of its body literals cannot be proved.

3. Current approaches

This section offers a deeper insight into the representation approaches outlined in the introduction (TPR, GPR and LPR), as well as the respective tools.

3.1. TPR: Tree proof representation

Explanations that adopt TPR take the form of a tree, a data structure most users are familiar with. In TPR, the (top-down or bottom-up) path from root to leaf represents the various inference steps, after which the final conclusion is derived. Naturally, each distinct node in the path represents an individual inference step.

Fig. 1 illustrates a TPR-based proof visualization for the sample proof of $+\partial\neg payBonus(john)$ in Section 2.3. In this representation, tree nodes alternate between proofs of literals and rule labels.

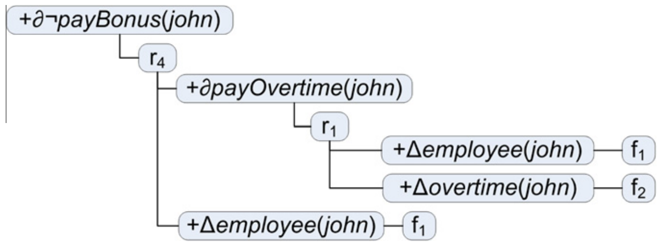


Fig. 1. TPR proof visualization.

Because of the structural similarities and hierarchical nature met in trees and XML, TPR is considered highly suitable for visualizing XML-based explanations, like those based on *PML* [46] and *PXS* [7].

TPR is popular among Prolog systems. For instance, *SWI-Prolog* [50] includes a tree-based graphical debugger, which provides a visual trace of a program. The user can select any node in the tree, in order to examine the context of that node. Graphical debugging is enabled via the guitracer command, which launches the graphical front-end of the application. The debugger window consists of various panes that display current bindings at run-time, diagrammatic traces of the call history as well as highlighted source code listings.

The *Visual Prolog Development Environment* [14] also contains a similar tree-based debugger that can follow program execution and observe the program state. The debugger is activated through the GUI menu and can execute multiple tasks, like: display memory and stack views, show trap points and backtrack points, display class and object facts along with their values, perform step into, step over and visualize fail and exit.

The *Execution Tree Viewer*, the tool for graphic visualization of a running *Prolog IV* program, is another paradigm [17]. The software offers a full view of an execution tree, a view of a proof, rapid access to a given point in the execution tree and help in rerunning the execution up to a given point. The main window of the software displays the execution tree of a program that comprises an AND/OR tree with graphical information about the level of each AND node. The tree is intended to be seen as a three dimensional tree display in perspective and consists of two types of nodes: *call nodes*, represented as colored call-boxes that include the predicate name, and *rule nodes*, represented as small, white rule-boxes that match the call to the head of a rule.

3.2. GPR: Graphical proof representation

GPR proofs are displayed as directed graphs, where nodes may represent premises and conclusions of rules, while rules can be represented either as edges or as nodes. The flow of the graph follows the $(premises) \rightarrow (conclusions)$ course. Enhancing the graph with further types of edges and/or connections can result in displaying more complex logics, e.g. Nute's *d-graphs* [38]. Fig. 2 illustrates a GPR-based proof visualization for $+\partial\neg payBonus(john)$ from Section 2.3.

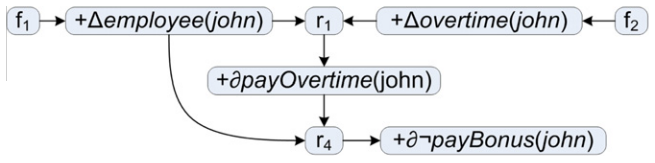


Fig. 2. GPR proof visualization.

A GPR example is *ProbeIt!* [43], a general-purpose visualization tool for visualizing both logical proofs generated by inference engines as well as workflow execution traces. The explanations are rendered as directed acyclic graphs (DAGs), but are encoded via the *Proof Markup Language* (*PML*) [46], an OWL-based language for representing justifications of computationally derived results. Probe-It! consists of three primary views to accommodate the visualization of different kinds of provenance information: results, justifications, and provenance, which refer to final and intermediate data, descriptions of the generation process (i.e., execution traces) and information about the sources, respectively.

Another GPR-based tool presented in [5], is dedicated to visualizing proofs produced from inference performed on defeasible logic rule bases by *DR-Prolog* [12]. The tool applies our *DRVgraph* visualization framework [28], which visualizes defeasible logic rule sets as directed acyclic graphs that feature distinct node and connection types. However, the specific tool applies DRVgraph for a fundamentally different purpose, which involves visualizing defeasible logic proofs instead (see Section 4.4 for a deeper insight into the differences among the two approaches). A further interesting feature is the tool's verbalization module that verbalizes explanations and makes clear which rule(s) support which conclusion(s), as well as other defeasible logic specifics, like superiority relationships.

3.3. LPR: Logical proof representation

LPR proofs offer a textual proof representation, based on some type of logic formalism. Often an LPR-based trace of the program execution can provide a sufficient basis, on which to build an explanation facility and generate explanations in a user-understandable language. For instance, the proof of $+\partial\neg payBonus(john)$ from Section 2.3 could be presented in a *CLIPS*-like [20] trace fashion in the following form:

```
==> f1: employee(john)
==> f2: overtime(john)
FIRE r1: f1, f2
==> f3: payOvertime(john)
FIRE r4: f1, f3
==> f4 : ¬payBonus(john)
```

All facts, even derived ones, receive IDs, which are used in justifying rule activations and derivations of conclusions. Prolog and expert systems typically apply LPR in representing inference explanations. Justifications stem from AND/OR trees created during reasoning that allow the system to explain its conclusions and reasoning process.

An example with historical value is *MYCIN* [19], a well-known system for providing consultation in establishing the proper diagnosis and therapy for patients with infectious disease problems. The system presents justifications for the derived conclusions in a natural-language format, accompanied with certainty factors for each derivation. However, MYCINs explanation facilities displayed the drawback of containing implicit knowledge regarding the diagnostic tasks and this knowledge was inaccessible to the explanation system. An improved version of the software, called *NEOMYCIN* [47], promised to make this implicit knowledge explicit and show the impact that this reconfiguration of knowledge has on generating explanations.

A drastically different approach is presented in [8], where the authors attack the explanation issue through an agent-based architecture, promising knowledge reusability, modularity and high quality explanations. The justifications generated by the agents are text-based, following a MYCIN-like format (see above) and revolve around "why" questions. Interestingly, the explanations also contain hyperlinks that enhance the understandability and interaction of the generated text. Hyperlinks in the generated HTML change automatically as the system's knowledge about new terms increases. As its authors suggest, the specific architecture is open and scalable and, thus, new services, such as "why not", "what if", etc. could be imported in the future.

A more modern and Semantic Web-related paradigm is a framework for developing knowledge-centric Clinical Decision Support Systems (CDSSs) [27]. The framework performs knowledge modeling through a synergy between multiple ontologies (domain ontology, Clinical Practice Guidelines-CPG ontology, patient ontology) and consists of three modules: rule-authoring, execution and justification trace sub-module. The latter generates a text-based justification trace of the rule execution, to assist medical practitioners in understanding the rationale behind the inferred recommendations. The justification trace initiates with the derived facts (an inferred patient recommendation) and generates facts, which serve as premises for deriving the patient recommendation, recursively.

3.4. Comparison

Table 1 displays a qualitative comparison of the TPR, GPR and LPR proof presentation approaches. The inclusion of certain classes of features (interaction, performance, clarity), as well as the respective members (e.g. scalability, comprehensibility, etc.) is based on relevant surveys (e.g. [26,9], while the final class of defeasible logic features is based on our personal experience and intuition on using and teaching defeasible logic concepts and comprises an essential comparison coefficient. The qualitative assessment regarding each characteristic (i.e. "*Yes*"/"*No*", "*High*"/"*Medium*"/"*Low*", etc.) is based on inherent properties of the approaches as well as traits of the corresponding tools implementing these

Table 1
Qualitative comparison of proof visualization approaches.

	TPR	GPR	LPR
Interaction			
Zoom & Pan	No	Yes	No
Collapse/expand	Yes	No	No
Animations	No	Yes	No
Performance			
Scalability	High	Low	High
Information overload	Low	High	Medium
Clarity			
Interrelationships	Low	High	Low
Readability/Comprehensibility	High	Medium	Medium
Perception & Cognition	Medium	High	Medium
Defeasible logic specifics			
Rule type support	Low	High	Medium
Rule superiority representation	Low	High	Low
Support for blocked rules	Low	High	Medium

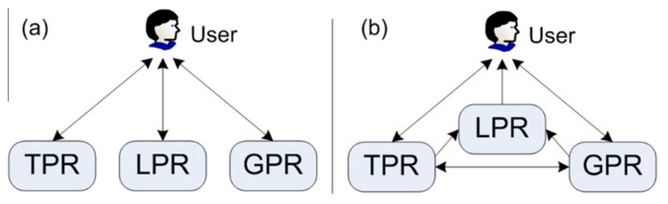
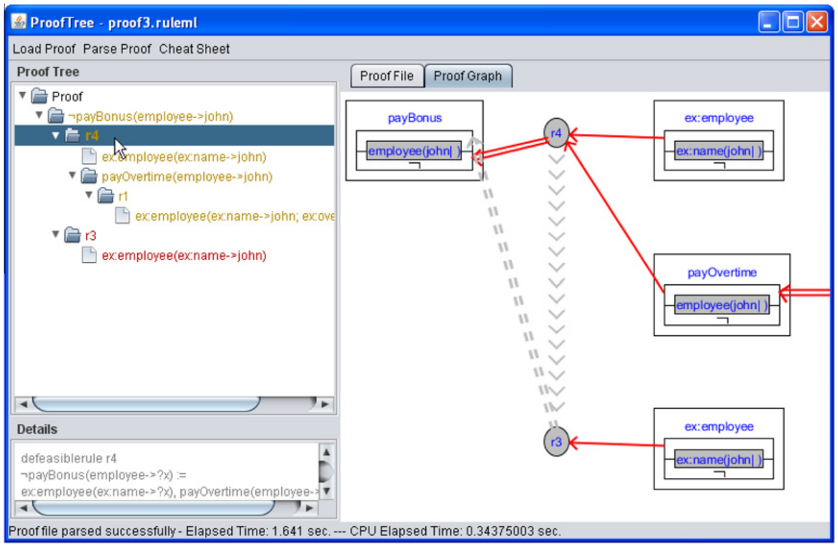


Fig. 3. (a) Common approaches typically feature a variety of isolated representations. (b) Our hybrid approach features three intercommunicating representations.



Proof
¬payBonus(employee->john)
r4
ex:employee(ex:name->john)
payOvertime(employee->john)
r1
ex:employee(ex:name->john; ex:ove
r3
ex:employee(ex:name->john)

defeasiblerule r4
¬payBonus(employee->?x) :=
ex:employee(ex:name->?x), payOvertime(employee->

Proof file parsed successfully - Elapsed Time: 1.641 sec. --- CPU Elapsed Time: 0.34375003 sec.

Fig. 4. VProof$_H$ main window.

approaches. It is possible that there are exceptions in the literature (e.g. an LPR that offers zoom & pan functionality), but the table aims at describing the characteristics of the majority of the tools implementing the three approaches).

As expected, TPR and LPR perform better as far as scalability and information overload are concerned, while GPR offers superior interaction. Regarding clarity, TPR and LPR generate a more comprehensible representation, since users are overall more familiarized with tree and logic-based representations, contrary to application-specific graph-based representations, generated by GPR. On the other hand, GPR can display more clearly all the underlying interrelationships among the various elements (e.g. different rule types, superiority relations, etc. – see Section 2.1 and Fig. 4, since it offers a greater variety of interconnections. On the contrary, TPRs connections among tree nodes simply denote an "implies" correlation.

Finally, as for representing defeasible logics specifics, like superiority relationships and rule types (see Section 2.1), GPR seems considerably more efficient, since it can feature a variety of node types. It is also important to note that the size of a defeasible logic rule base is not the only important factor regarding complexity. Contrary to monotonic logic rule bases, a conclusion derived from a defeasible logic rule base is not always evident to justify, even if the rule base is relatively small, due to multiple unforeseen rule attacks and counter-attacks. Nevertheless, GPR by itself is not sufficient for representing defeasible logic proofs, since the comprehensibility as well as the performance superiority offered by the other two approaches are nontrivial features that should not be absent in a proof visualization tool.

Conclusively, although the elimination of the corresponding drawbacks is not completely plausible, the proposed hybrid approach (described subsequently) ensures the combination of the dominant advantages from each representation, aiming at an overall superior representation.

4. Our approach: Hybrid proof visualization

Although the tools described above offer a user-centred view of a proof trace, they also display the disadvantage of relying on a specific representation approach, which requires a degree of adjustment to the tool. This fact potentially aggravates the learning curve and confuses the user. Besides the disadvantages presented previously (Section 3.4), this is a further drawback that the proposed hybrid approach attempts to eliminate, by combining all three aforementioned representations and bringing together TPR, GPR and LPR into a single visualization.

Nevertheless, as a feature, the combination of all approaches is not adequate in itself. An additional but equally important quality is the interaction and interoperability between the different representations. Thus, contrary to other approaches (Fig. 3(a)), like, e.g. *IWBrowse* [35], which offers a variety of isolated representations (natural-language-like, graphical, KIF-based [24], our hybrid approach presents the user with the interplay of three simultaneous complementary representations. The use of multiple simultaneous visualizations is a great way to help users explore, discover and reason and, possibly, confirm their intuitions [40].

More specifically, a proof visualization via the hybrid approach consists always of three parts, one for each of the three representations. The TPR and GPR parts are highly interactive, allowing the user to traverse the proof tree and graph respectively, while user-intervention (e.g. selection of a specific tree or graph node) in one of the two representations is reflected on the other representation as well. LPR is by nature less interactive, since it is text-based, but it is not static either, since user-triggered modifications in TPR and GPR are reflected in LPR as well. The overall functionality of the proposed hybrid approach is illustrated in Fig. 3(b).

4.1. VProofH – Overview

VProofH is a software tool for visualizing defeasible logic proofs that adopts the hybrid approach. The main window of the tool's GUI is displayed in Fig. 4.

The figure illustrates all available proof representations: On the top-left is the TPR, on the right is the corresponding GPR and on the bottom-left is a POSL-based LPR for each selected tree/graph element, so that it is not necessary for users to remember the details for each rule. Finally, a status bar at the bottom displays useful information in the form of messages.

The overall functionality of the software is displayed in Fig. 5. A proof file is loaded through the GUI and is parsed by the *Proof Par-*

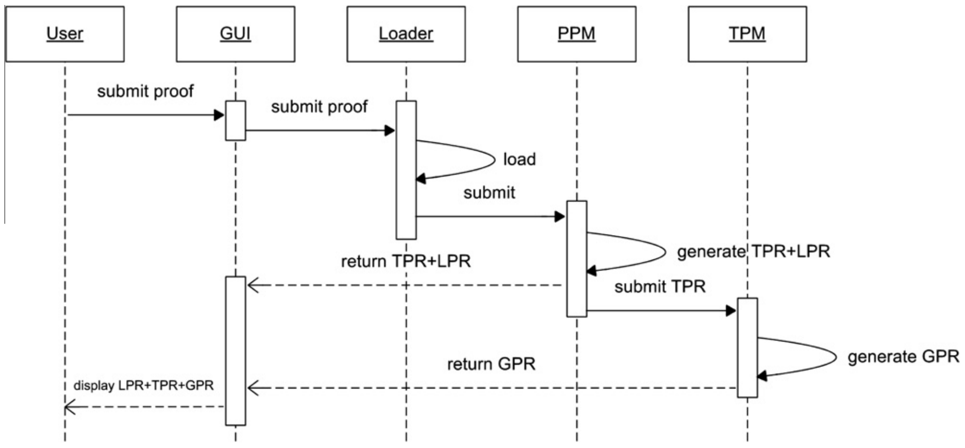


Fig. 5. VProof$_H$ overall functionality.

```
<RuleML rdf_import="http://.../ex1.rdf" rdf_export="export.rdf" rdf_export_classes="payBonus">
    .........
    <Implies ruletype="defeasiblerule">
        <oid><Ind uri="http://.../ex1.ruleml#r4"> r4</Ind></oid>
        <head>
            <Neg>
                <Atom>
                    <op><Rel>payBonus</Rel></op>
                    <slot><Ind>employee</Ind><Var>x</Var></slot>
                </Atom>
            </Neg>
        </head>
        <body>
            <And>
                <Atom>
                    <op><Rel uri="ex:employee"/></op>
                    <slot><Ind uri="ex:name"/><Var>x</Var></slot>
                </Atom>
                <Atom>
                    <op><Rel>payOvertime</Rel></op>
                    <slot><Ind>employee</Ind><Var>x</Var></slot>
                </Atom>
            </And>
        </body>
        <superior><Ind uri="http://.../ex1.ruleml#r3"/></superior>
    </Implies>
    .........
</RuleML>
```

Fig. 6. Rule base fragment in the DR-RuleML syntax.

ser Module (*PPM*), which generates the TPR and LPR (see Section 4.3). The TPR is then fed to the *Tree Parser Module* (*TPM*), which analyzes the proof tree structure and generates the GPR (see Section 4.4). All three representations are eventually presented to the user via the GUI.

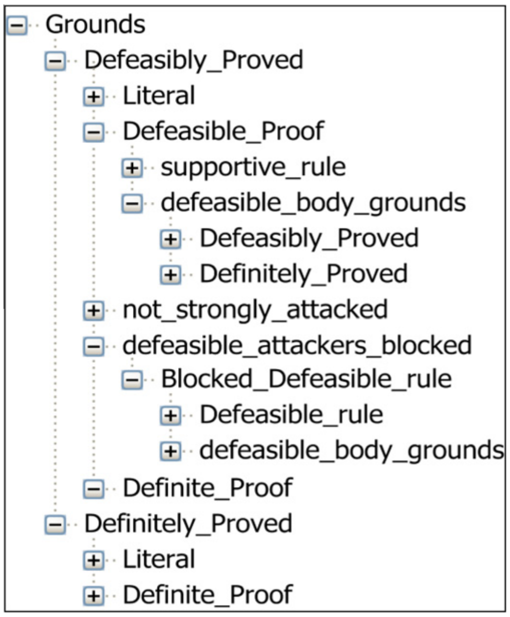
4.2. Proof XML Schema (PXS)

As mentioned, VProofH visualizes proofs generated by DR-DEVICE [6]. The rule language of DR-DEVICE, called DR-RuleML, is an OO RuleML [15] (v. 0.91) extension that deals with defeasible logic and the system's CLIPS implementation. The choice of OO RuleML coincides with the philosophy behind the reasoning engine; DR-DEVICE literals follow an object-oriented structure, where the predicate name is a class name and the arguments are named slots (or attributes). The engine employs an OO RDF data model, where properties are treated as normal encapsulated attributes of resource objects. The predicate name of an atomic proposition corresponds to the type of an RDF resource and the slot arguments to resource properties. Fig. 6 demonstrates a fragment (rule r_4) from the sample rule base in Section 2.3.

The *Proof XML Schema* (*PXS*) for DR-DEVICE proof trace explanations is a further extension to RuleML that is built on-top of DR-RuleML and comprises an extended version of previous work of ours [7]. The top-level element of the proof schema is the Grounds element that consists of multiple rule conclusions, proved and non-proved. As outlined in Section 2.2, conclusion proofs can either be definite or defeasible and are encapsulated inside schema elements Definitely_Proved and Defeasibly_Proved, respectively.

Definitely proved literals consist of the literal itself, encapsulated by a Literal element, as well as the definite proof tree, contained in a Definite_Proof element that explains why the literal is definitely provable. A literal is definitely proved if there is a strict clause (i.e. a strict rule or a fact), whose body literals are also definitely proved. The literal can be a positive atom or its negation, or even a reference to an RDF resource (notice that DR-DEVICE uses RDF resources as facts and its conclusions are also materialized as RDF resources).

On the other hand, defeasible proofs are more complex and require a supportive rule (contained inside a supportive_rule element) that can either be a strict or defeasible rule, whose body literals are defeasibly proved. The explanation for the supportive



```
⊟ Grounds
    ⊟ Defeasibly_Proved
        ⊞ Literal
        ⊟ Defeasible_Proof
            ⊞ supportive_rule
            ⊟ defeasible_body_grounds
                ⊞ Defeasibly_Proved
                ⊞ Definitely_Proved
        ⊞ not_strongly_attacked
        ⊟ defeasible_attackers_blocked
            ⊟ Blocked_Defeasible_rule
                ⊞ Defeasible_rule
                ⊞ defeasible_body_grounds
        ⊟ Definite_Proof
    ⊟ Definitely_Proved
        ⊞ Literal
        ⊞ Definite_Proof
```

Fig. 7. High-level hierarchical overview of PXS (recursive elements are not expanded further).

rule can be found inside the defeasible_body_grounds element, which is a sibling element to supportive_rule and, in turn, consists of further Definitely_Proved and Defeasibly_Proved elements.

The defeasible conclusion must not be strongly attacked, i.e. its negation must not be definitely proved. Rules that defeasibly attack the current one must all be blocked, so that the defeasible conclusion of this rule prevails. A defeasible rule (or a defeater) is blocked, either when its body literals are not defeasibly proved (enclosed inside a not_defeasible_body_grounds element), or when it is attacked by a superior defeasible rule (enclosed inside an Attacked_by_Superior element), whose body literals are defeasibly proved. A strict rule is blocked, if its body literals are not definitely proved (indicated by a not_definite_body_grounds element). Finally, inferior defeasible rules are considered as blocked.

Not proved conclusions follow a similar structure, i.e. the supportive rule that could not prove something must be included along with the reason why this happened. In the case of a defeasible non-proof (element Blocked_Defeasible_rule nested inside a defeasible_attackers_blocked element), reasons include either the non-proof of (some of) the body literals or a definitely proved negated literal or an undefeated defeasible attacker. A defeasible attacker can be a defeasible rule or a defeater, whose body literals are proved and whose possible attackers have been blocked. Notice that, in order for a conclusion to be not defeasibly provable, it also must be not definitely provable (Section 2.2), similarly to the blocked strict rule case. Fig. 7 displays a high level hierarchical overview of the basic PXS elements.

Overall, the aim is to provide portable and sharable proofs, where all references inside justifications can be also represented by URIs. Thus, rules can either be in-lined in the proof tree or there can be an external reference to rules in another RuleML document. Similarly, proofs for body literals can either be encapsulated in the proof tree of the rule head or can be referenced from another document location. Fig. 8 displays a proof document fragment, represented in the syntax of the DR-RuleML PXS extension.

4.3. Proof Parser Module (PPM)

As seen previously, PXS is highly recursive, therefore, PPM follows this standard, being a top-down PXS parser, consisting of several recursive procedures. The respective functions for retrieving a defeasible proof for a literal are listed below. The corresponding functions for the other types of (non)proofs are similar and are omitted here due to space limitations.

```
<Grounds>
    <Defeasibly_Proved>
        <Literal><Neg><Atom>
            <op><Rel>payBonus</Rel></op>
            <slot><Ind>employee</Ind><Ind>john</Ind></slot>
        </Atom></Neg></Literal>
        <Defeasible_Proof>
            <supportive_rule>
                <Defeasible_rule ruletype="defeasiblerule">
                    <oid><Ind uri="http://.../ex1.ruleml#r4">r4</Ind></oid>
                    .................
                </Defeasible_rule>
            </supportive_rule>
            <defeasible_body_grounds>
                <Definitely_Proved>
                    <Literal><Atom>
                        <op><Rel uri="ex:employee"/></op>
                        <slot><Ind uri="ex:name"/><Ind>john</Ind></slot>
                        <slot><Ind uri="ex:overtime"/><Ind>true</Ind></slot>
                        <slot><Ind uri="ex:newEmployee"/><Ind>false</Ind></slot>
                    </Atom></Literal>
                    <Definite_Proof>.................</Definite_Proof>
                </Definitely_Proved>
                <Defeasibly_Proved>
                    <Literal><Atom>
                        <op><Rel>payOvertime</Rel></op>
                        <slot><Ind>employee</Ind><Ind>john</Ind></slot>
                    </Atom></Literal>
                    <Defeasible_Proof>.................</Defeasible_Proof>
                </Defeasibly_Proved>
            </defeasible_body_grounds>
            <not_strongly_attacked/>
            <defeasible_attackers_blocked>.................</defeasible_attackers_blocked>
        </Defeasible_Proof>
    </Defeasibly_Proved>
</Grounds>
```

Fig. 8. Proof document fragment.

```
Algorithm parseProof
  Input: Tree structure ($t$)
  Output: Updated tree structure ($t'$)
1. $n \leftarrow$ getXpath("//Defeasibly_Proved/Grounds")
2. if $n \neq \emptyset$ then /* (the proof is defeasible) */
3.   $t' \leftarrow$ parseDefeasiblyProved($n$, $t$)
4. else /* $n = \emptyset$ (the proof is definite) */
5.       $t' \leftarrow$ parseDefinitelyProved(getXpath("//
   Definitely_Proved/Grounds"), $t$)
6.   return $t'$
Function parseDefeasiblyProved
  Input: Current XML tree node ($n$), Tree structure ($t$)
  Output: Updated tree structure ($t'$)
1. $t \leftarrow$ insert(getXpath("//"+n+"/Literal/Atom"), $t$)
2. $p \leftarrow$ getXpath("./Defeasible_Proof")
3. if $p \neq \emptyset$ then
4.   $t' \leftarrow$ parseDefeasibleProof($p$, $t$)
5. else /* $p = \emptyset$ */
6.   $t' \leftarrow$ parseDefiniteProof(getXpath("//"+n+"/
   Definite_Proof"), $t$)
7. return $t'$
Function parseDefeasibleProof
  Input: Current XML tree node ($n$), Tree structure ($t$)
  Output: Updated tree structure ($t'$)
  1. $t \leftarrow$ insert(getXpath("//"+n+"/supportive_rule/
  Defeasible_rule"))
2. $g \leftarrow$ getXpath("//"+n+"/defeasible_body_grounds/
   Defeasibly_Proved")
3. if $g \neq \emptyset$ then
4.   $t' \leftarrow$ parseDefeasiblyProved($g$, $t$)
5. else /* $g = \emptyset$ */
6.   $t' \leftarrow$ parseDefinitelyProved(getXpath(
     "//"+n+"/defeasible_body_grounds/Definitely_Proved"), $t$)
7. $t' \leftarrow$ parseBlocked(getXpath("//"+n+"/Blocked"), $t'$)
8. return $t'$
Function parseBlocked
  Input: Current XML tree node ($n$), Tree structure ($t$)
  Output: Updated tree structure ($t$)
1. $r \leftarrow$ getXpath("//"+n+"/Blocked_Defeasible_rule")
2. if $r \neq \emptyset$ then
3.   $t' \leftarrow$ insert(getXpath("//"+n+"/Defeasible_rule"))
4.   $t' \leftarrow$ parseDefeasiblyProved(getXpath(
     "//"+n+"/defeasible_body_grounds/
     Defeasibly_Proved"), $t'$)
5. else /* $r = \emptyset$ */
6.   $t' \leftarrow$ insert(getXpath("//"+n+"/Strict_rule"))
7.   $t' \leftarrow$ parseDefinitelyProved(getXpath(
     "//"+n+"/definite_body_grounds/Definitely_Proved"), $t'$)
8. return $t'$
```

Function *getXpath* accepts as argument an XPath expression, which is evaluated and returns the result node(s). Furthermore, *insert* is a function for appending a new node into the tree structure. The flow of the algorithm gradually investigates each "layer" of (defeasible or definite) proof, which in turn consists of further, "nested" proof layers and so on. For each definite proof, the concluded literal as well as the proof itself is retrieved, while for each defeasible proof, it is necessary to retrieve the supportive rule as well (see also previous subsection). Additionally, the parsing algorithm detects all (defeasible or strict) blocked rules, namely the rules, whose prerequisites could not be proved or those (defeasible) rules that were attacked by superior defeasible rules.

As can be observed, certain pseudo-code parts are unavoidably similar, since all functions refer to top-down tree navigation during parsing. Thus, since trees are self-similar structures, processing of different nodes in different levels should also be performed by similar procedures. On the other hand, different levels in the tree represent nodes with dissimilar semantics and should be treated differently by the parser.

The proof trace is visualized in a TPR similar to Fig. 1, but with extra features for capturing defeasible logic characteristics. These features include node coloring, for indicating the proof status (green = definitely proved, yellow = defeasibly proved, red = not proved) and custom node font size, for indicating rule superiority (among two rule nodes at the same level, the node with the notably biggest font size represents the superior rule). Both features are illustrated in Fig. 4.

4.4. Tree Parser Module (TPM)

The TPM is fed with the tree structure (TPR) created by the proof parser (PPM) and generates a corresponding GPR. Similarly to the tool presented in [5], VProof$_H$ also deploys the *DRVgraph* rule visualization schema proposed in [28], but a newer version of it, based on *Piccolo2D* [41]. *Piccolo2D* is a toolkit that supports the development of 2D structured graphics programs and especially *Zoom-able User Interfaces* (*ZUI*s). DRVgraph provides a "graphical language" (like e.g. in [42] for rendering defeasible logic rule bases as digraphs and is more thoroughly presented subsequently. A VProof$_H$ GPR example is displayed in Fig. 4.

The parsing of the TPR by TPM is rather straightforward: tree nodes are transformed into DRVgraph elements, which are then placed into layers in the graph, according to an algorithm that visually stratifies graphs consisting of DRVgraph elements. The algorithm shares some basic similarities with the one presented in previous work of ours [29], where a prior algorithm is presented for visual stratification of defeasible logic rule sets (and not proofs). More specifically, the proof graph also consists of DRVgraph elements, but its structure is quite different from a static rule base, simply because it must intuitively represent a single run of the rule base on a specific conclusion instance. This means that (a) many of the elements of the rule base might not be included, since they do not contribute to the specific proof, and (b) the class and slot patterns of the DRVgraph elements (see next subsection) do not contain variables, but constants, since the proof refers to a specific conclusion and not a general rule base.

According to the algorithm, graph elements are placed in strata (i.e. columns), with the first stratum located on the utmost left and the graph following a left-to-right orientation to the flow of information. Since each tree node holds information regarding the other nodes it is connected to as well as the type of the connection (e.g. derivation), drawing the connections in the graph does not present a challenge for TPM.

4.5. The GPR of VProofH – DRVgraph

DRVgraph[1] [28,30] is a defeasible logic rule base visualization framework based on Nute's *defeasible logic graphs* (*d-graphs*) [38]. However, the framework adopts a variety of additional features that enhance the expressive power of the graph. DRVgraph is based on *directed graphs* (*digraphs*) for visually representing defeasible logic rules. In an attempt to leverage the inability of (directed) graphs to associate data of a variety of types with the nodes and edges in

[1] Available at: http://lpis.csd.auth.gr/systems/drvgraph/index.html.

the graph, the use of distinct node and connection types is proposed. Thus, a graph in the DRVgraph framework consists of the following elements:

- *Class Boxes*: They serve as containers and are populated with one or more class patterns. Since the proof schema is based on OO RuleML, each class box corresponds to a class and is labeled with the respective class name.
- *Class Patterns*: They express conditions on (filtered) instance subsets of a specific class. Each class pattern consists of two adjacent atomic formula boxes, with the upper one representing a positive and the lower one representing a negated atomic formula. Positive atomic formulas represent subsets of objects of the class that have been proven to satisfy the condition of the class pattern, while negative atomic formulas represent objects that have been proven to NOT satisfy the condition.
- *Slot Patterns*: They express queries via named slots (rather than positional arguments) on class instances. Each slot pattern is identified by a slot name and can also contain condition patterns. The argument list of each slot pattern is divided in two parts, separated by "|"; on the left all the variables are placed and on the right all the corresponding expressions and conditions, regarding the variables on the left. In the case of constant values, only the left-hand side is utilized;
- *Rule Circles*: They represent rules and are identified by the rule label.
- *Edges*: There exist five types of connections in the graph: three for the rule type (strict rule, defeasible rule, defeater), one for the superiority relationship, and one simple arrow connection type for connecting the class patterns of rule bodies to the rule circles. This results in the different graph elements being represented more distinctively.

More details regarding DRVgraph can be found in [28,30]. As already stated, although the DRVgraph framework was initially designed for visualizing defeasible logic rule bases, in this work, as well as in the work by Avguleas et al. [5], it is determined that the framework is also suitable for representing defeasible logic proofs as well, without posing the need to modify the fundamental structure of its elements. What are fundamentally different in this paper are the placement and the content of the elements, since they need to represent proofs of specific conclusions, rather than universally quantified rules.

An example of the above is seen in Fig. 9, where rule r_1 from Section 2.3 is displayed, following the DRVgraph scheme. The "employee" class box contains two class patterns applied on the *employee* class, each of which contains one slot pattern. The "payOvertime" class box corresponds to the *payOvertime* class (i.e. all people who pay FPOS) and contains a single class pattern. As can be observed, the argument list of each slot pattern is divided in two parts, separated by "|"; on the left all the variables are placed and on the right all the corresponding expressions and conditions, regarding the variables on the left. In the case of constant values, only the left-hand side is utilized; thus, the second class pattern of the "employee" class box in Fig. 9, for example, refers to all overtime employees. This way the content of the slot arguments is clearly depicted and easily comprehended. A sample of the DRVgraph representation from within the VProof$_H$ tool is illustrated in Fig. 4 – right-hand side panel in the main window of the software.

4.6. The LPR of VProofH – POSL and d-POSL

VProof$_H$'s LPR uses *POSL* (positional-slotted language) [16] an ASCII language that integrates Prolog's positional and F-logic's

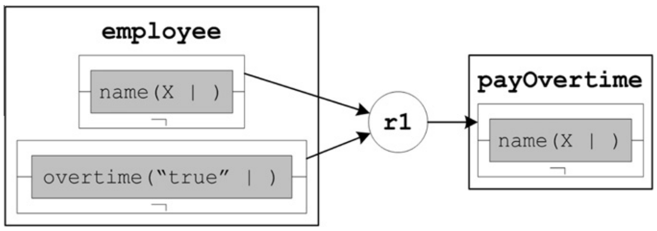


Fig. 9. DRVgraph representation of a rule.

slotted syntaxes for representing knowledge (facts and rules) in the Semantic Web. POSL is primarily designed for human consumption, since it is faster to write and easier to read than any XML-based syntax, with a variety of POSL-based applications already beginning to emerge (e.g. [21,13]. In essence, VProof$_H$ utilizes an extension to POSL, called *d-POSL*, which handles the specifics of defeasible logics and comprises a further novel contribution included in this work.

POSL adopts a Prolog-like syntax that offers the option of constructing atoms in a positional and/or slotted fashion. Thus, an atom in POSL has the following form:

$p(r_1 \to f_1; r_2 \to f_2; \ldots; r_L \to f_L; e_1, e_2, \ldots, e_M; r_{L+1} \to f_{L+1}; r_{L+2} \to f_{L+2}; \ldots; r_N \to f_N)$

For example, rule "The discount for a customer buying a product is 5% if the customer is premium and the product is regular" can be expressed in positional POSL as:

```
discount(?customer, ?product, percent5):-
  premium(?customer), regular(?product).
```

and in slotted POSL as:

```
discount(cust->?customer; prod->?product;
rebate->percent5):-
  premium(cust->?customer), regular(prod->?product).
```

Variables in POSL are denoted with a preceding "?". A deeper insight into POSL, its unification scheme, the underlying webizing process (i.e. the introduction of URIs as names in a system to scale it to the Web – orthogonal to the positional/slotted distinction), and its typing conventions along with examples is found in (Boley, 2004).

d-POSL maintains all the critical components of POSL, extending the language with elements that are essential in defeasible logics:

- *Rule type:* Similarly to *d-Prolog* [37], binary infix functors are introduced (":-", ":=", ":~") to denote the rule type ("*strict*", "*defeasible*", "*defeater*", respectively)
- *Rule label:* d-POSL employs a mechanism for applying rule labels that satisfy the need to express superiorities among rules in defeasible logic.
- *Conflicting literals:* Conflicting literals in d-POSL are represented as headless rules (actually, integrity constraints), whose bodies consist of conflicting literal pairs as well as the conditions between the corresponding arguments that ensure that the literals are not unifiable.
- *Negation & negation-as-failure:* Since defeasible logics require both types of negation (see Section 2.1), d-POSL represents classical negation as ¬/NOT and negation-as-failure as ~/NAF.

For example, the rule base in Section 2.3 would become:

```
employee(name->john; overtime->‘‘true’’).
r1: payOvertime(name->X) := employee(name->X;
  overtime->‘‘true’’).
r2 : ¬payOvertime(name- > X) :=
employee(name->X; overtime->‘‘true’’;
  newEmployee->‘‘true’’).
r3: payBonus(name- >X) := employee(name->X).
r4 : ¬payBonus(name- > X) :=
employee(name->X), payOvertime(name->X).
r4 > r3.
```

An EBNF specification of d-POSL grammar is given in the Appendix.

5. Scalability testing

In order to determine VProof$_H$'s scalability regarding the size of the proof, *DTScale* was used, a tool that accompanies *Deimos*, a query answering defeasible logic system [33]. DTScale was used for generating scalable defeasible logic theories for testing, which were then translated to DR-RuleML and loaded into DR-DEVICE. The latter performed inference and produced the results, also generating the respective PXS-compliant proof files that were loaded into VProof$_H$ and the respective CPU loading times were measured.

VProof$_H$'s performance was then compared to that of a similar tool, which is the system presented in [5], also mentioned before, in Section 3.2. To the best of our knowledge, the specific application is the only software that visualizes defeasible logic proofs along with the proposed VProof$_H$. Nevertheless, the two systems differ significantly, as far as the underlying reasoning engine is concerned. The other tool is designed for use with the DR-Prolog defeasible reasoner [12], while VProof$_H$ works with proofs generated by the DR-DEVICE defeasible reasoning engine [6]. DR-Prolog operates in a backward chaining fashion and produces non-provability proof chains due to the unsatisfaction of rule conditions, while DR-DEVICE works in a forward chaining fashion and cannot produce non-provability proof chains based on unsatisfiability, but only non-provability proof chains that are based on rule conflicts. The advantage is that proofs can be produced faster. Furthermore, the most cumbersome feature of defeasible logic typically is rule conflicts and not unsatisfiability (which also exists in classical deductive logic); therefore, users usually have most trouble understanding rule conflicts when tracing rule execution. Consequently, while the input data (formal proof representations) is produced by different systems, which are based on different reasoning techniques (forward vs. backward chaining), the only common point among the two visualizing tools is the graph-based visual representation schema (DRVgraph – see Section 4.5).

In order to perform the comparison, PXS proof files were translated into DR-Prolog proofs [12] and were then loaded into the other system. The latter was chosen for the comparison, since it is functionally very similar to VProof$_H$: both systems visualize defeasible logic proofs, both proof formats are RuleML-based and both tools utilize the DRVgraph framework (see Section 4.4). Nevertheless, VProof$_H$ comprises a more advanced solution, as it features a variety of representation approaches, contrary to the other tool's sole GPR-based visualization. The tests were performed on a P4 PC (3.2 GHz) with 2 GBs main memory.

DTScale can generate various types of theories, for exploring the numerous aspects of defeasible logic operational semantics. For example, *chain* theories of size *n* start with a fact a_0 and continue with a chain of *n* defeasible rules of the form $a_{i-1} \Rightarrow a_i$. A defeasible proof of a_n will use all of the rules and the fact. More details about the various theory types can be found in [33]. For each tested theory, it was ensured that all included rules and facts were used. Our tests included the following test theory types: *chain*, *dag*, *levels*, *teams* and *tree*. The other two types featured by DTScale, *circle* and *mix*, were omitted, because the former causes cyclic execution in DR-DEVICE and the latter was considered redundant. Table 2 displays the *order*, i.e. total number of nodes (rules + atoms), and *size*, i.e. total number of edges (including superiority edges) of the generated GPR, depending on the loaded defeasible theory type.

Results (Table 3) show that both systems can successfully render most (even large-scale) proofs in a few seconds, with the exception of voluminous teams theories. Comparatively, VProof$_H$ performs much better, offering at the same time a broader range of proof trace views (tree, graph, d-POSL) and functionality (zoom/pan, graph navigation etc.). VProof$_H$'s superior performance

Table 2
Order and size of proof graphs.

Theory	Rules	Atoms	Priorities	Order	Size
Chain(r)	r	$r+1$	0	$2r+1$	$2r$
Levels(r)	$4r+5$	$2r+3$	$r+1$	$6r+8$	$6r+7$
Teams(r)	$4\sum_{i=0}^{r}4^i$	$4\sum_{i=0}^{r-1}4^i$	$2\sum_{i=0}^{r-1}4^i$	$4\sum_{i=0}^{r}4^i+4\sum_{i=0}^{r-1}4^i$	$8\sum_{i=0}^{r}4^i+2\sum_{i=0}^{r-1}4^i$
Tree(r, k)[a]	$\sum_{i=0}^{r-1}k^i$	$(k+1)\sum_{i=0}^{r-1}k^i$	*0*	$(k+2)\sum_{i=0}^{r-1}k^i$	$\sum_{i=0}^{r-2}k^i$
DAG(r, k)[b]	$rk+1$	$r(k+1)$	0	$(2k+1)r+1$	$3(rk+1)$

[a] A tree(r, k) theory features a k-branching tree of depth r, where every literal occurs once.
[b] A drag(r, k) theory features a k-branching tree of depth r, here every literal occurs k times.

Table 3
Scalability comparison. Time is measured in seconds.

System	VProof$_H$				[5]			
RB size	10	100	500	1000	10	100	500	1000
Chain	.036	.172	.773	1.963	.261	.372	1.087	2.794
Levels	.075	.349	1.796	5.433	.433	.992	3.122	7.027
Teams	5.880	11.413	–	–	8.396	17.263	–	–
Tree (k = 2)	.398	1.221	6.119	17.866	1.390	3.477	12.409	–
DAG (k = 2)	.067	.211	1.156	3.023	.427	.695	2.634	5.613

Table 4
Descriptions and objectives of tasks in the user evaluation.

Task	Task description	Task objective
1	*Estimate number of rules*: Make an estimation regarding the total number of rules involved in the proof. Answers were given in ranges, e.g. [1, 10], [11, 20], [21, 30], [31, 40]	How correctly does the user estimate the total number of rules in the proof?
2	*Estimate number of nodes*: Make an estimation regarding the proof order (total number of nodes, i.e. rules+literals, involved in the proof). Answers were given in ranges, e.g. [1, 25], [26, 50], [51, 75], [76, 100]	How correctly does the user estimate the order of the proof?
3	*Determine rule types*: Determine the rule types (strict, defeasible, defeater) involved in the proof	Does the user perceive all types of rules that are involved in the proof?
4	*Determine type of specific rule*: Detect a specified rule and determine its type	Can the user easily detect the specified rule? Does he/she successfully comprehend its rule type?
5	*Determine superiority pairs*: Specify whether any superiority relationships exist in the proof and, if so, determine the corresponding rule pairs, indicating each time the superior and inferior rules	Can the user easily detect existing superiority relationships in the proof? Can he/she successfully indicate the superior and inferior rules?
6	*Determine proof status*: Determine the proof status of a specified literal (i.e. strictly/defeasibly proved/not-proved)	Can the user correctly detect the inference steps leading to the specified literal? Can he/she realize the types of the rules involved?
7	*Determine blocking factors of a rule*: Detect a specified rule and determine why the activation of the rule was blocked	Can the user easily detect the specified rule? Does he/she successfully comprehend the reasons why activation of the rule was blocked?
8	*Give a description of a specified rule*: Give a description of the specified rule in some logical formalism	Can the user correctly grasp the logical content of a specified rule? Do the given representations assist adequately?

is due to the system's improved algorithm efficiency and the other tool's resource-consuming verbalization module.

6. User evaluation

A task-oriented user evaluation of VProof$_H$ was performed, in order to assess whether the three simultaneous representations (TPR, GPR, LPR) are indeed more helpful than having a sole representation approach (e.g. only GPR). Additionally, the evaluation was expected to identify possible flaws and deficiencies both in the representation methodologies, as well as the software itself, and provide valuable feedback for improvements.

Generally, task-oriented evaluations for various Semantic Web applications are quite popular (e.g. [48,39]. Our user evaluation involved 17 test users, mostly postgraduate students attending the Semantic Web course at our university, as well as research personnel at our Lab. The involvement of less knowledgeable users in the evaluation process would probably require their long-term preparation on issues related to Semantic Web and Logic Programming, a factor that would reduce our ability of checking VProof$_H$'s intuitiveness and usefulness related to defeasible logic issues alone. A future goal, nevertheless, is to conduct a wider user evaluation that will span other categories of (less and more knowledgeable) users as well, like developers or knowledge engineers, as these are described in the introduction. The test users were given a short introductory lecture on the basics of defeasible logics and were then seated in front of a PC, in order to evaluate the software. They were given three (3) groups of tasks; each task group involved a different rule base and consisted of eight tasks. The corresponding tasks in each group were similar, but each group required the use of a different combination of representation approaches. Thus, Task Group 1 (TG 1) involved using TPR and LPR only (the subset of VProof$_H$ called VProof$_T$), Task Group 2 (TG 2) involved using GPR and LPR (subset VProof$_G$), while Task Group 3 (TG 3) allowed the users to utilize all three representations (TPR, GPR and LPR), namely the full-fledged hybrid representation VProof$_H$ (isolating LPR would not lead to any significant conclusions, since it comprises a secondary representation approach that better assists in displaying single items of knowledge and not sets of facts or rules). Also, in order to eliminate the effect of learning, the task groups were given to each participant in a random order. Table 4 displays the list of tasks, accompanied by a respective objective for each task.

Search facilities in all versions of the software were disabled, so that users would rely each time only on the given representation for discovering the specific elements required by the tasks. Fig. 10 illustrates the results of the user evaluation.

The column chart in the figure shows the median times required for completing each task; the median was used for eliminating the effect of outliers. As can be observed, the hybrid approach represented by VProof$_H$ performed better in all tasks, assisting the users in completing the tasks notably sooner than the other two representations. Regarding the other two subsets, it is interesting to note that VProof$_T$ performs better in navigation-centered tasks (like detecting a specific rule – task #4), while VProof$_G$ proves superior in tasks regarding the estimation of a proof size, as well as the comprehension of the underlying proof theory. Thus, a secondary conclusion that was derived from the evaluation involved the utility of our DRVgraph representation against the more traditional tree-based approaches.

As far as the comparison of correct answers among the task groups is concerned, the superiority of VProof$_H$ in almost all cases is clearly demonstrated (stacked column chart in Fig. 11), with the minor exception of tasks #1 and #2, where the graph-based representation performs slightly better. Especially in the second task, none of the three approaches performed adequately, probably because of the nature and complexity of the proofs given to test users, which made it more difficult to estimate the total number of nodes in each proof. It is obvious in both tasks, nevertheless, that the approaches including GPR (VProof$_G$ and VProof$_H$) greatly outperform the tree-based approach, as far as assessments regarding numbers of nodes and proof sizes are concerned.

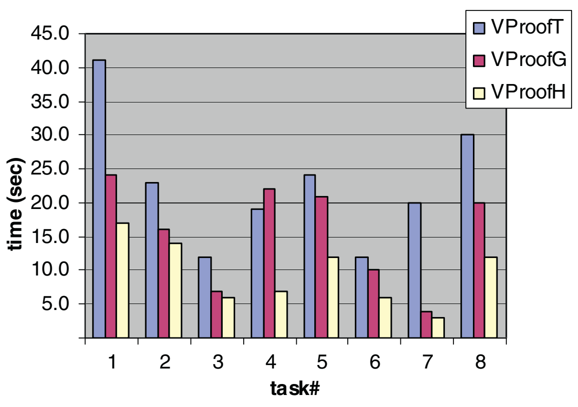


Fig. 10. Time (in s) needed to complete tasks.

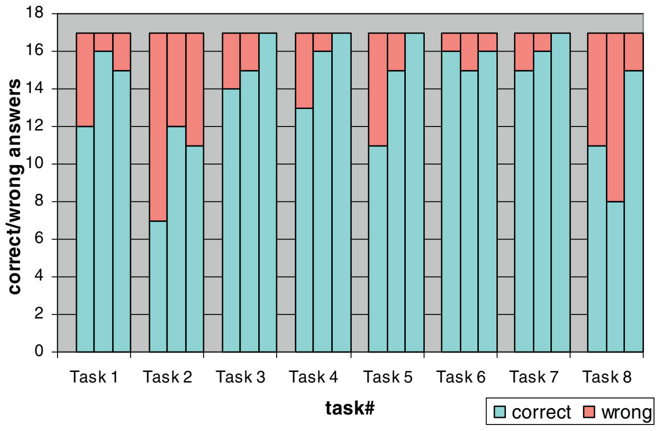


Fig. 11. Comparison of correct answers among the three task groups.

Overall, it is primarily concluded that the hybrid representation approach improves the user's understanding of a proof and, in some cases, significantly boosts the performance of the other two isolated representations (e.g. task #8). It is also evident that our DRVgraph-based GPR typically remains more effective than TPR, as was previously stated.

In order to assess, whether the differences in times and correctness are statistically significant, a one-way ANOVA was conducted on the median times for completing each task as well as the corresponding number of correct answers. The null hypothesis (H_0) was that there are no differences among the three approaches VProof$_T$, VProof$_G$ and VProof$_H$. The results of ANOVA displayed in Table 5 indeed reveal a statistically significant effect of the representation approach used ($p \leqslant 0.05$), thus the null hypothesis is rejected. It is therefore concluded that there is a significant difference among *at least* one pair of representation approaches, without excluding the possibility that the differences among *all* pairs are eventually statistically significant.

Since there is a significant difference among deploying the different approaches, an independent-samples *t*-test was conducted to investigate the difference among using any two approaches.

The results in Table 6 show that the time taken to accomplish the tasks by using the combined hybrid approach (VProof$_H$) is significantly less than using the tree-only approach (VProof$_T$). However, the difference between using VProof$_T$ vs. VProof$_G$ is less significant, as is also the difference between VProof$_G$ vs. VProof$_H$, although the latter is marginally concluded ($p = 0.085$). The slight difference among VProof$_G$ and VProof$_H$ probably suggests that appending TPR to VProof$_G$ (namely, adding the tree facility over the graph-based GPR that results in VProof$_H$) adds marginally significant value to the representation.

Table 5
ANOVA results of median times for completing the designated tasks among different representations.

	Sum of squares	Degrees of freedom	Mean square	*F*	Significance
Between approaches	678.083	2	339.042	5.925	0.009
Within approaches	1201.750	21	57.226		
Total	1879.833	23			

Table 6
Independent-samples *t*-test results of median times for completing the designated tasks among the different representations.

	Significance
Between VProof$_T$ and Vproof$_G$	0.120
Between VProof$_T$ and Vproof$_H$	0.004
Between VProof$_G$ and Vproof$_H$	0.085

Table 7
Paired-samples *t*-test results of task correctness among the different representations.

	Significance
Between VProof$_T$ and Vproof$_G$	0.117
Between VProof$_T$ and Vproof$_H$	0.001
Between VProof$_G$ and Vproof$_H$	0.134

Similar conclusions are derived when examining the statistical significance of task correctness. Table 7 displays the results of paired-samples t-test among any two approaches. The task correctness of the hybrid approach is significantly better than the TPR-based approach ($p = 0.001$), while the difference between the other two pairs of approaches is less significant ($p = 0.117$ and $p = 0.134$, respectively). It is therefore concluded, that the hybrid approach does not significantly improve a user's ability to successfully complete tasks, when compared to the VProof$_G$ approach that features a GPR-based representation.

Conclusively, the user evaluation indicates that there is statistically significant difference among the hybrid approach and the tree-based TPR as far as both task completion time and correctness are concerned. However, the tests comparing the GPR-based approaches vs. the hybrid approach suggest that the latter does not add statistically significant value to performance and correctness, although the significance regarding the difference in completion times was only marginally rejected. Consequently, VProof$_H$ definitely assisted users in completing the designated tasks sooner than the other two approaches, while its performance regarding task correctness is considered, if not better, then at least equivalent to GPR. The overall improvement of VProof$_H$ against the other, "lighter" approaches is, therefore, considered notable.

7. Conclusions and future work

The paper argued that logic, proof and trust comprise the target of upcoming research efforts in the Semantic Web and pointed out the need for proof and explanation visualization mechanisms that are addressed to human users for encouraging them to trust system answers. A hybrid proof visualization approach was presented, which, contrary to other methodologies that offer isolated representations, offers multiple simultaneous views of an execution trace, combining tree, graph and textual visualizations for representing proofs. Additionally, a software tool called *VProof$_H$* was implemented, which efficiently visualizes defeasible logic proofs, offering a variety of interoperating representation and visualization approaches that adapt to user needs. The representations are interactive, meaning that user-triggered modifications in one of them are reflected in the rest of the representations as well.

The user evaluation of VProof$_H$ revealed certain deficiencies of the software and directions for future improvements. For instance, an interesting idea would be to explicitly represent the superiority relationship rule pairs in a separate list, which is a feature almost all test users pointed out. On the other hand, an appealing development would be tightly integrating VProof$_H$ with one or more defeasible logic reasoners (e.g. DR-DEVICE, DR-PROLOG, SPINdle – see Section 1), for offering an explanation visualization facility to the respective human users. Finally, the underlying DRVgraph model should be extended for visualizing more defeasible logic elements, like conflicting literals.

As a final remark, it should be noted that, since proofs depend on statements whose truth cannot be safely assumed, the process of merely rendering machine-oriented proofs into a human-comprehensible format is not adequate by itself. Therefore, proofs must be augmented by trust. A mechanism towards this direction is to enable digital signatures of proof documents [4]; digital signatures can provide a universal basis for determining the level of trust featured by each Semantic Web proof document. Additionally, since all the facts that are contained in a proof could be assigned to a source, users need to retrieve some information about the sources in order to trust them [1]. Towards this direction, applications that rate the authoritativeness of sources could be developed.

Appendix A

EBNF description of d-POSL grammar:

```
(* a rulebase contains facts, rules, superiority relationships and
   conflicting literals *)
rulebase ::= (fact | rule | suprel | conf)*.
(* definition of facts *)
fact ::= fid “:” literal “.”.
literal ::= ”∼”? atom.
atom ::= pred “(“ sname “->” svalue (“;” Sname “->” svalue)* “)”.
(* definition of rules *)
rule ::= rid “:” head imp body? “.”.
head ::= literal .
imp ::= ”:-“ | ”:=” | ”:∼”.
body ::= cond_cluster ((“,”cond_cluster) | (“,”naf_cluster))* .
cond_cluster ::= literal ((“,” literal) | (“,” cond))* .
naf_cluster ::= ”\+” cond_cluster .
cond ::= unkeyed extfunc unkeyed .
(* definition of superiority relationships *)
suprel ::= rid “>” rid “.”.
(* definition of conflicting literals *)
conf ::= (“:-“ | ”:=”) literal (“,” literal) + (var “\=” var) + ”.”.
(* rest of definitions *)
unkeyed ::= svalue ::= ind | var.
var ::= “?” symbol.
fid ::= rid ::= extfunc ::= ind::= pred ::= sname ::= symbol.
```

References

[1] G. Antoniou, A. Bikakis, N. Dimaresis, M. Genetzakis, G. Georgalis, G. Governatori, E. Karouzaki, N. Kazepis, D. Kosmadakis, M. Kritsotakis, G. Lilis, A. Papadogiannakis, P. Pediaditis, C. Terzakis, R. Theodosaki, D. Zeginis, Proof explanation for a nonmonotonic Semantic Web rules language, Data and Knowledge Engineering 64 (3) (2008) 662–687.

[2] G. Antoniou, D. Billington, G. Governatori, M.J. Maher, Representation results for defeasible logic, ACM Transactions on Computational Logic 2 (2) (2001) 255–287.

[3] G. Antoniou, M.J. Maher, D. Billington, Defeasible logic versus logic programming without negation as failure, Journal of Logic Programming 41 (1) (2000) 45–57.

[4] D. Artz, Y. Gil, A survey of trust in computer science and the Semantic Web, Web Semantics: Science, Services and Agents on the World Wide Web, 5(2) (2007) 58–71.

[5] I. Avguleas, K. Gkirtzou, S. Triantafilou, A. Bikakis, G. Antoniou, E. Kontopoulos, N. Bassiliades, Visualization of proofs in defeasible logic, in: 2008 International Symposium on Rule Interchange and Applications (RuleML-2008), LNCS, 5321, Orlando, Florida, USA, Springer, 2008, pp. 197–210.

[6] N. Bassiliades, G. Antoniou, I. Vlahavas, A defeasible logic reasoner for the Semantic Web, International Journal on Semantic Web and Information Systems 2 (1) (2006) 1–41.

[7] N. Bassiliades, G. Antoniou, G. Governatori, Proof explanation in the DR-DEVICE system, in: 1st Int. Conference on Web Reasoning and Rule Systems (RR 2007), LNCS, 4524, Austria, Springer-Verlag, 2007, pp. 249–258.

[8] S.R. El-Beltagy, A.A. Rafea, A.H. Sameh, An agent based approach to expert system explanation, in: A.N. Kumar, I. Russell, (Eds.), Twelfth Int. Florida Artificial Intelligence Research Society Conference, AAAI Press, 1999, pp. 153–159.

[9] C. Bennett, J. Ryall, L. Spalteholz, A. Gooch, The aesthetics of graph visualization, in: International Symposium on Computational Aesthetics in Graphics, Visualization, and Imaging, Banff, Alberta, Canada, June 20–22, 2006, pp. 57–64.

[10] T. Berners-Lee, J. Hendler, O. Lassila, The Semantic Web, Scientific American 284 (5) (2001) 34–43.

[11] T. Berners-Lee, W. Hall, J.A. Hendler, K. O'Hara, N. Shadbolt, D.J. Weitzner, A framework for web science, Foundations and Trends in Web Science 1 (1) (2006) 1–130.

[12] A. Bikakis, C. Papatheodorou, G. Antoniou, The DR-prolog tool suite for defeasible reasoning and proof explanation in the Semantic Web, in: 5th Hellenic Conference on Artificial Intelligence: Theories, Models and Applications, LNAI, 5138, Syros, Greece, Springer-Verlag, 2008, pp. 345–351.

[13] Y. Biletskiy, G.R. Ranganathan, An invertebrate semantic/software application development framework for knowledge-based systems, Knowledge-Based Systems 21 (5) (2008) 371–376.

[14] T.W. de Boer, A Beginners Guide to Visual Prolog. <http://www.download.pdc.dk/vip/72/books/deBoer/VisualPrologBeginners.pdf>.

[15] H. Boley, Object-oriented RuleML: user-level roles, URI grounded clauses, and order-sorted terms, in: Rules and Rule Markup Languages for the Semantic Web (RuleML-2003), Sanibel Island, Florida, LNCS 2876, Springer-Verlag, 2003.

[16] H. Boley, POSL: an integrated positional-slotted language for Semantic Web knowledge. <http://www.ruleml.org/submission/ruleml-shortation.html>.

[17] P. Bouvier, Visual tools to debug prolog IV programs, in: P. Deransart, M.V. Hermenegildo, J. Maluszynski, (Eds.), Analysis and Visualization Tools For Constraint Programming, Constraint Debugging (DiSCiPl project), LNCS 1870, Springer-Verlag, 2000, pp. 177–190.

[18] D. Bryant, P. Krause, A review of current defeasible reasoning implementations, The Knowledge Engineering Review 23 (3) (2008) 227–260.

[19] B.G. Buchanan, E.H. Shortliffe, Rule based expert systems: The mycin experiments of the Stanford Heuristic Programming Project, in: The Addison-Wesley Series in Artificial Intelligence, Addison-Wesley Longman Publishing Co., Inc., Boston, MA, 1984.

[20] CLIPS 6.3 Reference Manual, Volume I, Basic Programming Guide. <http://www.clipsrules.sourceforge.net/documentation/v630/bpg.pdf>.

[21] B.L. Craig, The OO jDREW Engine of Rule Responder: Naf Hornlog RuleML Query Answering, in: A. Paschke, Y. Biletskiy, (Eds.), 2007 International Conference on Advances in Rule Interchange and Applications, LNCS, Springer-Verlag, Berlin, Heidelberg, 2007, pp. 149–154.

[22] C.G. Fernandes, V. Furtado, A. Glass, D.L. McGuinness, Towards the generation of explanations for Semantic Web Services in OWL-S, in: Proceedings of the 2008 ACM Symposium on Applied Computing (SAC'08), ACM, New York, NY, 2008, pp. 2350–2351.

[23] M. Gandhe, T. Finin, B. Grosof, SweetJess: Translating DamlRuleML to Jess, in: Int. Workshop on Rule Markup Languages for Business Rules on the Semantic Web (in conjunction with ISWC 2002), Sardinia, Italy, 2002.

[24] M. Genesereth, R. Fikes, Knowledge Interchange Format Version 3.0 Reference Manual, Technical Report, Logic Group, Comp. Sci. Dept., Stanford Univ., 1992.

[25] Q. Guo, M. Zhang, Question answering based on pervasive agent ontology and Semantic Web, Knowledge-Based Systems 22 (6) (2009) 443–448.

[26] I. Herman, G. Melancon, M.S. Marshall, Graph visualization and navigation in information visualization: a survey, IEEE Transactions on Visualization and Computer Graphics 6 (1) (2000) 24–43.

[27] S. Hussain, S.R. Abidi, S.S.R. Abidi, Semantic Web framework for knowledge-centric clinical decision support systems, in: Proceedings of 11th Conference on Artificial Intelligence in Medicine (AIME2007), LNCS, vol. 4594, Springer, Berlin, 2007, pp. 451-455.

[28] E. Kontopoulos, N. Bassiliades, G. Antoniou, Visualizing defeasible logic rules for the Semantic Web, in: 1st Asian Semantic Web Conference (ASWC'06), LNCS, 4185, Beijing, China, 3–7 September 2006, Springer-Verlag, pp. 278–292.

[29] E. Kontopoulos, N. Bassiliades, G. Antoniou, Visual stratification of defeasible logic rule bases, in: 19th IEEE international conference on tools with artificial intelligence (ICTAI'07), IEEE, Patras, Greece, 2007, pp. 238–245.

[30] E. Kontopoulos, N. Bassiliades, G. Antoniou, Deploying defeasible logic rule bases for the Semantic Web, Data and Knowledge Engineering 66 (1) (2008) 116–146.

[31] H.P. Lam, G. Governatori, The making of SPINdle, in: International Symposium on Rule Interchange and Applications (RuleML'09), 5858, 2009, pp. 315–322.

[32] S.A. Ludwig, Comparison of a deductive database with a Semantic Web reasoning engine, Knowledge-Based Systems 23 (6) (2010) 634–642.

[33] M.J. Maher, A. Rock, G. Antoniou, D. Billington, T. Miller, Efficient defeasible reasoning systems, International Journal of Tools with Artificial Intelligence 10 (4) (2001) 483–501.

[34] D.L. McGuinness, P.P. da Silva, R. Fikes, J. Jenkins, G. Frank, Inference web: portable and sharable explanations for question answering, in: AAAI Spring Symposium Workshop on New Directions for Question Answering, Stanford University, 2003.

[35] D.L. McGuinness, P.P. da Silva, Infrastructure for Web explanations, in: D. Fensel, K. Sycara, J. Mylopoulos (Eds.), 2nd Int. Semantic Web Conference (ISWC'03), LNCS, 2870, Springer, 2003, pp. 113–129.

[36] D. Nute, Defeasible logic, in: D.M. Gabbay, C.J. Hogger, J.A. Robinson (Eds.), Handbook of Logic in Artificial Intelligence and Logic Programming, vol. 3, Oxford University Press, 1994, pp. 353–395.

[37] D. Nute, Defeasible prolog, in: M. Covington, D. Nute, A. Vellino, (Eds.), Prolog Programming in Depth, second ed., Prentice-Hall, Upper Saddle River, NJ, 1997, pp. 345–405.

[38] D. Nute, K. Erk, Defeasible logic graphs: I. Theory, Decision Support Systems 22 (3) (1998) 277–293.

[39] H. Oliver, G. Diallo, E. de Quincey, D. Alexopoulou, B. Habermann, P. Kostkova, M. Schroeder, S. Jupp, K. Khelif, R. Stevens, G. Jawaheer, G. Madle, A user-centred evaluation framework for the sealife Semantic Web browsers, BMC Bioinformatics 10(Suppl. 10) (2009) S14+.

[40] D. Petrelli, S. Mazumdar, A. Dadzie, F. Ciravegna, Multi visualization and dynamic query for effective exploration of semantic data, in: A. Bernstein, D.R. Karger, T. Heath, L. Feigenbaum, D. Maynard, E. Motta, K. Thirunarayan (Eds.), 8th International Semantic Web Conference (ISWC'09), Chantilly, VA, USA, LNCS, 5823, Springer, 2009, pp. 505–520.

[41] Piccolo2D – A Structured 2D Graphics Framework. <http://www.piccolo2d.org>.

[42] E. Pulvermueller, S. Feja, A. Speck, Developer-friendly verification of process-based systems, in: H. Fujita (Ed.), Knowledge-Based Systems Special issue on "Intelligent Formal Techniques for Software Design: IFTSD", vol. 23(7), 2010, pp. 667–676.

[43] N. Del Rio, P.P. da Silva, Probe-it! Visualization Support for Provenance, in: 2nd International Symposium on Visual Computing (ISVC 2), Lake Tahoe, NV, USA, LNCS, 4842, Springer, 2007, pp. 732–741.

[44] T.R. Roth-Berghofer, M.M. Richter, On explanation, Künstliche Intelligenz 22 (2) (2008) 5–7.

[45] P. Shvaiko, F. Giunchiglia, P.P. da Silva, D.L. McGuinness, Web explanations for semantic heterogeneity discovery, in: Proceedings of the 2nd European Semantic Web Conference (ESWC 2005), 2005, pp. 303–317.

[46] P.P. da Silva, D.L. McGuinness, R. Fikes, A proof markup language for Semantic Web services, Information Systems 31 (4–5) (2006) 381–395.

[47] J. Sotos, MYCIN and NEOMYCIN: two approaches to generating explanations in rule based expert systems, Aviation, Space, and Environmental Medicine 61 (1990) 950–954.

[48] Y. Sure, V. Iosif, First results of a Semantic Web technologies evaluation, in: Common Industry Program at the Federated Event Co-Locating the Three International Conferences: DOA/ODBASE/CoopIS'02, University of California, Irvine, 2002, pp. 69–78.

[49] W. Swartout, S.W. Smoliar, Explanation: source of guidance for knowledge representation, in: K. Morik (Ed.), Knowledge Representation and Organization in Machine Learning, Springer Lecture Notes in Artificial Intelligence, vol. 347, Springer-Verlag, 1989, pp. 1–16.

[50] J. Wielemaker, An overview of the SWI-prolog programming environment, in: F. Mesnard, A. Serebenik, (Eds.), 13th International Workshop on Logic Programming Environments, Heverlee, Belgium, Katholieke Universiteit Leuven, 2003, pp. 1–16.

[51] B.A. Wooley, Explanation component of software system, ACM Crossroads 5 (1) (1998) 24–28.