EXTENDING A MULTI-AGENT REASONING INTEROPERABILITY FRAMEWORK WITH SERVICES FOR THE SEMANTIC WEB LOGIC AND PROOF LAYERS

Kalliopi Kravari¹, Konstantinos Papatheodorou², Grigoris Antoniou² and Nick Bassiliades¹

¹Dept. of Informatics, Aristotle University of Thessaloniki, Greece
²Institute of Computer Science, FORTH, Greece
Overview

Motivation

EMERALD
  - Reasoners

DR-Prolog Reasoner
  - RuleMLParser
  - RDFParser
  - ResultParser

Defeasible Proofing Services
  - Defeasible logic
  - Defeasible Proof Generator Service
  - Proof Validator Service

Conclusions – Future Work
Motivation

SW evolution

fundamental SW technologies established/standardized

Researchers currently focus on logic and proofs

No common:
- rule/logic representation formalism
- W3C standards
  - e.g. RIF → W3C standard/no wide adoption

Standards BUT not universal languages:
- e.g. RIF, RuleML
  - Share syntactic features
  - Differ in semantics
Motivation

Agents should somehow share an understanding of each other’s position

Solution A: equipping each agent with inference/reasoning mechanism
  • agents would require common interchange language
  • language translations

Solution B: wrapping reasoning services as IAs
  • no need for common logic/rule paradigm
    inferencing tasks conducted by the reasoning services.

Trust:
users should trust systems
→ systems should explain their actions, sources, and beliefs

Traditional trust models guarantee agents’ trustworthiness but not the correctness of the inference service itself

Need for automating proof generation, exchange and validation
Overview

➤ Motivation

➤ **EMERALD**
  - Reasoners

➤ **DR-Prolog Reasoner**
  - RuleMLParser
  - RDFParser
  - ResultParser

➤ **Defeasible Proofing Services**
  - Defeasible logic
  - Defeasible Proof Generator Service
  - Proof Validator Service

➤ Conclusions – Future Work
EMERALD
A Multi-Agent Knowledge-Based Framework
Reasoners

- Built as agents
- Act as like web services
- Provide the reasoning services
- Launch an associated reasoning engine

**Reasoner:**
- *stands* by for new requests
- *gets* a valid request $\rightarrow$ *launches* the reasoning service $\rightarrow$ *returns* the results

**EMERALD** supports some reasoning engines that use a variety of logics,

- **DR-DEVICE, SPINdle, R-DEVICE** and **PROVA**
- defeasible logic, datalog-like, prolog-like rules
Overview

➢ Motivation

➢ EMERALD
  o Reasoners

➢ DR-Prolog Reasoner
  o RuleMLParser
  o RDFParser
  o ResultParser

➢ Defeasible Proofing Services
  o Defeasible logic
  o Defeasible Proof Generator Service
  o Proof Validator Service

➢ Conclusions – Future Work
DR-Prolog Reasoner

DR-Prolog\(^1\):
- built on-top of Prolog
- uses defeasible logic rules, facts and ontologies
- supports all major SW standards (e.g. RDF, RDFS, OWL, RuleML)
- deals with monotonic and nonmonotonic rules, open and closed world assumption and reasoning with inconsistencies

DR-Prolog Reasoner:
- follows the EMERALD Reasoners' general functionality
- stands by for new requests
- gets a valid request \(\rightarrow\) launches DR-Prolog \(\rightarrow\) returns the results

➢ Need for some new intermediate steps
  - to process the receiving queries
  - to send back the appropriate answer in RDF format

\(^1\)Antoniou, G., Bikakis, A.: DR-Prolog: A System for Defeasible Reasoning with Rules and Ontologies on the SW. IEEE TKDE 19,2
DR-Prolog Reasoner

RuleMLParser: RuleML to DR-Prolog

RuleMLParser
- receives the RuleML file with the query
- extracts the DR-Prolog rules
- stores them in native DR-Prolog format

- processes the rules in the RuleML file, generating the corresponding DR-Prolog rules
- extracts the queries that are included in the RuleML query, indicating whether it is an “answer” or a “proof” query.

Example RuleML:
```
<Implies ruletype="defeasiblerule">
  <oid>r1</oid>
  <head>
    <Atom neg="no">
      <Rel>acceptable</Rel>
      <Slot type="var">X</Slot>
      <Slot type="var">Y</Slot>
      <Slot type="var">Z</Slot>
      <Slot type="var">W</Slot>
    </Atom>
  </head>
  <body>
    <part type="atom">
      <Rel>name</Rel>
      <Slot type="var">X</Slot>
    </part>
    <part type="atom">
      <Rel>price</Rel>
      <Slot type="var">X</Slot>
      <Slot type="var">Y</Slot>
    </part>
  </body>
</Implies>
```

Example DR-Prolog Rule:
```
defeasible(r1, 
  acceptable(X,Y,Z,W), 
  [name(X,X), price(X,Y), size(X,Z), gardenSize(X,W))].
```

Example DR-Prolog Query:
```
defeasibly(acceptable(X,Y,Z,W))
```
DR-Prolog Reasoner

RDFParser: RDF rule base to DR-Prolog facts

- Turning the initial RuleML query and rulebase into DR-Prolog is not enough
- The fact base has to be translated too

RDFParser

- Uses
  - The SW Knowledge Middleware,
  - A set of tools for SW (RDF) KBs
  - Parsing, storage, manipulation, querying
- Extract the RDF triples
- Turn them to Prolog facts
**ResultParser**

- receives
  - the initial query (in DR-Prolog)
  - the results (a prolog list)
- returns the query results in RDF

- the returned RDF results contain only the results that are required by the initial query and **not** the complete RB available information

```
<dr-device:acceptable rdf:about="#acceptable0">
  <dr-device:X>a3</dr-device:X>
  <dr-device:Y>350</dr-device:Y>
  <dr-device:Z>65</dr-device:Z>
  <dr-device:W>0</dr-device:W>
</dr-device:acceptable>
```
Overview

➤ Motivation

➤ EMERALD
  o Reasoners

➤ DR-Prolog Reasoner
  o RuleMLParser
  o RDFParser
  o ResultParser

➤ Defeasible Proofing Services
  o Defeasible logic
  o Defeasible Proof Generator Service
  o Proof Validator Service

➤ Conclusions – Future Work
Defeasible Logic Intro

- Facts e.g. student(Sofia)
- Strict Rules e.g. student(X) → person(X)
- Defeasible Rules e.g. person(X) ⇒ works(X)
- Priority Relation between rules
  e.g. \( r: \) person(X) ⇒ works(X)
  \( r': \) student(X) ⇒ ¬works(X)
  \( r' > r \)

Proof theory example:
- A literal \( q \) is defeasibly provable if:
  - supported by a rule whose premises are all defeasibly provable AND
  - \( ¬q \) is not definitely provable AND
  - each attacking rule is non-applicable or defeated by a superior counter-attacking rule
Defeasible Proofing Services

(SW) Proof Layer

- assumed to answer why agents should believe the results
- no W3C recommended technology
Defeasible Proofing Services
Defeasible Proof Generator Service

DR-Prolog Reasoner

--- EQUIPPED ---

proof generator → produces → proof explanations → using → XSB logic programming system

| interprete
| proof's trace output
| convert
| into a meaningful representation.

explanations
in defeasible logic
(positive/negative)
supports
Defeasible Proofing Services
Proof Validator Service

Implementation assumptions:
• The theory is the one given to the proof generator.
  ➢ Any given theory is accepted as valid without any checks.
• No checks are performed recursively.
  ➢ Any information is required in depth more than one a priori.
• Any knowledge facts given in the theory is considered to be definitely provable
  ➢ not taking into account statements present in the proof supporting it.
• The minimal information that will contribute to the proof checking process is required.
Defeasible Proofing Services
Defeasible Proof Validator Service

Data Structures

Proof Validator features

- theory structures \( KB \) holding
- strict rules
- defeasible rules
- facts
- rules hierarchy

proof deduction structures \( KB \) holding

any deduced information already stated by the proof and confirmed by the validator

Example: a rule that adds knowledge to the definite \( KB \) is the following

\[
\text{definitelyCheck}(X, \text{printOn}) \leftarrow \\
\text{factkb}(F), \text{memberchk}(X,F), \text{addDefinitely}(X).
\]

Facts

all given theory facts are added in the definite knowledge base.

\( \rightarrow \) any stated fact is by default considered by the proof validator as definitely proved.

Rules

not stated explicitly in the contents of the proof validation result

\( \rightarrow \) since they are a priori accepted and considered valid.
Defeasible Proofing Services
Proof Validator Service

Example:

r1: a ⇒ e. r2: b ⇒ e. r1 > r3. r2 > r4.
r3: c ⇒ ¬e. r4: d ⇒ ¬e. a. b. c. d.

A valid proof:

defeasibly(a), defeasibly(b),
defeasibly(c), defeasibly(d),
defeasibly(e, r2)

1. Is there any rule “r2” in the theory, with head equal to e? → Yes.
2. Is there any attacking rule? → Yes, r3, r4.
   2.1 Is r2 of higher priority than r3? → No.
      2.1.1 Are the conditions of r3 (i.e. c) defeasibly provable? → Yes.
      2.1.2 Is there any attacking rule of r3 (different from r2), which defeats r3? → Yes, r1, because its conditions are met and r1 > r3.
2.2 Is r2 of higher priority than r4? → Yes.
Overview

- Motivation

- EMERALD
  - Reasoners

- DR-Prolog Reasoner
  - RuleMLParser
  - RDFParser
  - ResultParser

- Defeasible Proofing Services
  - Defeasible logic
  - Defeasible Proof Generator Service
  - Proof Validator Service

- Conclusions – Future Work
Conclusions - Future Work

A. EMERALD
• a fully FIPA-compliant MAS developed on top of JADE
• proposes the use of trusted, independently-developed reasoning services (REASONERS)
  1. Can offer inferencing on a variety of logics
  2. Can be used for related services such as
     a) proof explanations on the inference results
     b) Proof validations on exchanged proofs

B. DR-Reasoner
➢ new types of logic embedded in new Reasoners
➢ new proofing services designed and embedded

In future:
• Integrate broader variety of reasoning and proof validation engines
• Integrate the generated proofs with trust mechanisms
EMERALD available at:
http://lpis.csd.auth.gr/systems/emerald

CS-566 Project available at:
http://www.csd.uoc.gr/~hy566/project2010.html

Thank you!
Any Questions?