I3-D3

Combining Rules and Ontologies. A survey.

Project title:	Reasoning on the Web with Rules and Semantics
Project acronym:	REWERSE
Project number:	IST-2004-506779
Project instrument:	EU FP6 Network of Excellence (NoE)
Project thematic priority:	Priority 2: Information Society Technologies (IST)
Document type:	D (deliverable)
Nature of document:	R (report)
Dissemination level:	PU (public)
Document number:	IST506779/Linköping/I3-D3/D/PU/b3
Responsible editors:	Jan Maluszynski
Reviewers:	Patrick Lambrix, Uwe Assmann
Contributing participants:	Heraklion, Linköping, Lisbon, Manchester, New-York
Contributing workpackages:	I1, I3
Contractual date of deliverable:	28 February 2005
Actual submission date:	03 March 2005

Abstract

We survey existing approaches to the problem of combining rule languages with ontology languages for the Semantic Web. The focus is on the languages based on logic and on the reasoning in such languages. The objective is to give a uniform view of the approaches, and to outline related research topics important for REWERSE.

Keyword List

ontologies, ontology languages, description logics, datalog, F-logic, non-monotonic reasoning

Project co-funded by the European Commission and the Swiss Federal Office for Education and Science within the Sixth Framework Programme.

Combining Rules and Ontologies. A survey.

Grigoris Antoniou[1] **and Carlos Viegas Damásio**[2] **and Benjamin Grosof**[3] **and Ian Horrocks**[4] **and Michael Kifer**[5] **and Jan Maluszynski**[6] **and Peter F. Patel-Schneider**[7]

[1] ICS-FORTH
Science and Technology Park Crete
Vassilika Vouton, P.O. Box 1385
GR-711 10 Heraklion, Crete, Greece

[2] Department of Computer Science
Universidade Nova de Lisboa
P-2829-516 Caparica, Portugal

[3] MIT Sloan School of Management
50, Memorial Drive
Cambridge, MA02142, USA

[4] School of Computer Science
University of Manchester
Oxford Road
Manchester, M13 9PL, United Kingdom

[5] Department of Computer Science
State University of New York
Stony Brook, New York 11794-4400, USA

[6] Department of Computer and Information Science,
Linköping University
S-581 83 Linköping, Sweden

[7] Network Data and Services Research Department,
Bell Laboratories
600 Mountain Ave., 2D-405
Murray Hill, NJ 07974, U.S.A.

03 March 2005

Abstract
We survey existing approaches to the problem of combining rule languages with ontology languages for the Semantic Web. The focus is on the languages based on logic and on the reasoning in such languages. The objective is to give a uniform view of the approaches, and to outline related research topics important for REWERSE.

Keyword List
ontologies, ontology languages, description logics, datalog, F-logic, non-monotonic reasoning

Contents

Acknowledgement

This document integrates material from many authors. The authors of the sections surveying specific approaches are indicated at the beginning of each such section.

The co-authors involved in REWERSE gratefully acknowledge contributions of Benjamin Grosof and Peter Patel-Schneider, who are not members of REWERSE.

We acknowledge discussions with Thomas Eiter on the contents of Section 6.2.3

The work of M. Kifer was supported in part by NSF grant CCR-0311512 and by U.S. Army Medical Research Institute under a subcontract through Brookhaven National Lab.

1 Introduction

The Semantic Web initiative of W3C aims at "extension of the current web in which information is given well-defined meaning, better enabling computers and people to work in cooperation."[BLHL01]. There seems to be a broad consensus in the Semantic Web community that the Semantic Web should include rules as well as ontologies. This is reflected the Semantic Web stack diagram given in Figure 1.

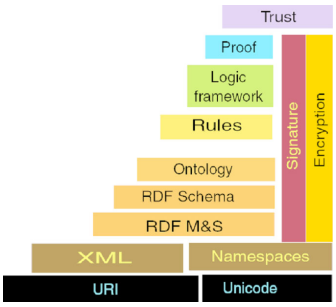

Figure 1: Semantic web stack diagram, from W3C.

At present the layers of the stack up to the ontology layer reached certain degree of maturity, reflected by the published W3C Recommendations, while integration of rules and ontologies is a subject of active research. The objective of this document is to provide a survey and a perspective of these developments.

The W3C view of ontologies is reflected by the Web Ontology Language OWL. Its core version OWL-DL is essentially an XML encoding of an expressive Description Logic, builds upon RDF and includes a substantial fragment of RDF-Schema, which itself can be seen as ontology language. The vocabularies defined in such ontologies consist of *classes* and *properties*, called also, respectively, *concepts* and *roles*. They have well-defined logical semantics, where they are considered, respectively, unary and binary predicates, and interpreted as relations.

The rules proposed for use in the Semantic Web must have clearly defined semantics. A natural choice concerns thus the classes of rules originating from logic programming and non-monotonic reasoning. They are based on different kinds of logics and have thus well defined declarative semantics, supported by well-developed reasoning algorithms. The simplest language of this kind consists of Horn clauses not including function symbols other than constants.

Integration of a rule language with an ontology language requires definition of a new language, its syntax and semantics and development of reasoning algorithms for the new language.

Existing proposals for integration of rule languages and ontology languages may be classified by the degree of integration:

- In the *hybrid* approach there is a strict separation between the ordinary predicates, which are basic rule predicates and ontology predicates, which are only used as constraints in rule antecedents. Reasoning is done by interfacing existing rule reasoner with existing ontology reasoner.

- In the *homogeneous* approach both ontologies and rules are embedded in a logical language $\mathcal{L}$ without making a priori distinction between the rule predicates and the ontology predicates. In this way a subset $\mathcal{R}$ of $\mathcal{L}$ is defined. For reasoning in $\mathcal{R}$ either a general reasoner of $\mathcal{L}$ can be reused, or a specialized reasoner for $\mathcal{R}$ is to be constructed.

An alternative to integration of given rule languages with given ontology languages may be the use of an expressive logic language where rules, ontologies and their combination can be expressed in a natural way.

The rest of this document is organized as follows. Section 2 sets the stage by providing a brief summary of the ontology languages and rule languages subject to integration efforts. We briefly discuss RDF, RDFS and OWL together with the Description Logic underlying OWL-DL. We sketch a general form of rules and some special cases, and outline their semantics.

The subsequent sections survey several integration efforts. Section 3 discusses integration of rules with RDF and RDFS. Section 4 and Section 5 are devoted to homogeneous approaches to integration of Description Logics with Horn rules. Section 6 surveys principles and examples of hybrid compositions. Embedding of rules and ontologies in F-logic is discussed in Section 7. Section 8 includes summary and discussion.

2 Preliminaries

This section gives an introduction to the formalisms for specifying ontologies proposed by W3C and to the rules used in logic programming and non-monotonic reasoning, which are subject of integration efforts discussed in the rest of this paper. We present first the Resource Description Framework (RDF) and its extension RDF Schema, which can be seen as simple ontology language. We then briefly outline basics of Description Logics, and we discuss the Description Logics underlying the W3C Web Ontology Language OWL. Finally we give a brief introduction to rule languages subject to the integration efforts.

2.1 Resource Description Framework and Schema

The Resource Description Framework [RDF] is a knowledge representation language for the Semantic Web. It is a simple assertional logical language which allows the specification of binary properties in the Semantic Web. The assertions, also called *triples*, are statements expressing that some resource (entity in the Semantic Web) is related to another entity or a value through a property. The resources and predicates are denoted by Uniform Resource Identifiers (URIs), and are the natural means to represent and share knowledge in the Semantic Web. The RDF has an intuitive graph model and semantics provided by a model theory inspired by first-order logic [Hay03], with appropriate notions of entailment.

The RDF Vocabulary Description Language (RDF Schema or RDFS) is a an extension of RDF providing a basic type system adapted for the Semantic Web context. It introduces the notions of class and property and provides mechanisms for specifying class hierarchies, property hierarchies and for defining domains and ranges of properties. A distinguished feature of the class model of RDF Schema is the separation of the intent of the class from its extent (the set of instances), allowing a class to be a member of its set of instances as well as application of properties to themselves. This freedom can be limited by semantic conditions, as it is done in the OWL Lite and OWL DL sublanguages of the Ontology Web Language [BvHH$^+$04]. For a discussion see [Hay03, BvHH$^+$04].

We briefly present the RDF language and its semantics, focusing on the aspects important for integration with rules. Full details can be found in the RDF suite of documents (see [MM04]).

2.1.1 RDF data model

This section introduces RDF data model and illustrates it by an example. An RDF assertion, called a triple, consists of a subject node s and an object node o, connected by a predicate p. Intuitively it expresses a statement that the relationship p holds between s and o, or in logical terms that $p(s, o)$ is

true. Sets of triples are represented as RDF graphs. In an RDF graph, nodes are labeled by URIs, values or blank nodes. URIs in nodes denote resources in the Semantic Web while the values can be either plain literals with optional language tags or typed literals. Blank nodes denote anonymous resources within a graph. Nodes are connected by directed edges from subject nodes to object nodes, labeled by URIs which identify predicates (or properties), and represent triples. The only restriction is that subject nodes cannot be labeled with literals, plain or typed. To summarize, an RDF graph is a set of triples $s\ p\ o$ where:

- s is the subject node, which can be either an URI or a blank node
- p is the predicate arc, a property identified by an URI
- o is the object node, which can be either an URI, a blank node or a literal.

In order to simplify presentation, we resort to the usual qualified name syntax $prefix : localname$ as shorthand for the URI obtained by concatenation of the namespace URI associated with $prefix$ with $localname$. The prefixes and associated namespaces URIs we use in this section are:

euro with namespace URI: `http://www.eurocup.org#`

rdf with namespace URI: `http://www.w3.org/1999/02/22-rdf-syntax-ns#`

rdfs with namespace URI: `http://www.w3.org/2000/01/rdf-schema#`

xsd with namespace URI: `http://www.w3.org/2001/XMLSchema#`

An example RDF graph can be found in Figure 2. This RDF graph states that, for instance, the resource identified by `<http://www.eurocup.org#groupA>` is related to the resource denoted by `<http://www.eurocup.org#Portugal>` via predicate `<http://www.eurocup.org#hasTeam>`. In the shortened notation we represent this statement (triple) as:

```
euro:groupA euro:hasTeam euro:Portugal .
```

The intended meaning of this statement is that Group A of Euro Cup has the team from Portugal; we should have similar triples for the other teams of the group. More interesting, is the triple `euro:Portugal rdf:type euro:Team` which states that the Portuguese team is a kind of `euro:Team`. The property `rdf:type` belongs to the RDF vocabulary and allows to express that some subject belongs to a kind of things, in this case to an Euro Cup team. The RDF Schema vocabulary designates these "kinds of things" as classes; RDF treats `rdf:type` triples as ordinary triples, no special meaning being assigned.

From the RDF graph we also know that `euro:Portugal` represents Portugal. The syntactic construct `"Portugal"` is a plain literal and denotes a value of the property `euro:represents` for resource `euro:Portugal`. Triples having literals in their object nodes represent attribute-value pairs.

The rest of the graph captures the facts that the Portugal team has some player (the blank node). This player is a LeftMidfielder and a RightMidfielder, his name is Luis Figo, wears number 7 and was born 1972-11-04. This part of the graph illustrates several interesting features of the Resource Description Framework. First, blank nodes denote anonymous resources and can be used freely in the graph, both at subject and object nodes. A resource might belong simultaneously to several classes. Furthermore, plain literals can have a language tag, specifying the language of the literal's text. Finally, typed literals can also be used by appending the URI which identifies the datatype of the literal, in this particular

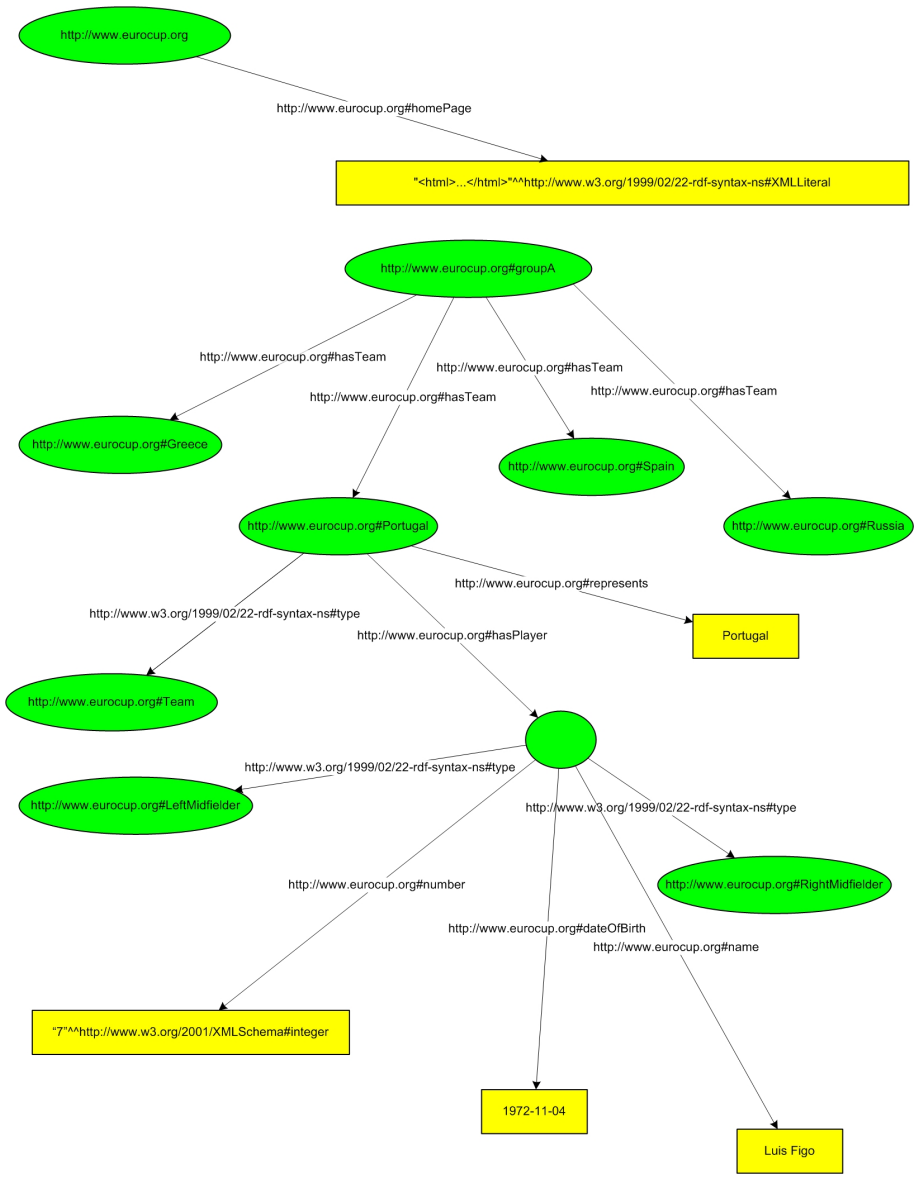

Figure 2: The Euro Cup example RDF graph

example, the XML Schema datatype integer. Notice also that XML data can be included, resorting to the RDF builtin datatype `rdf:XMLLiteral`.

There are several formats to serialize RDF graphs, the most important being the RDF/XML representation [Bec04], the N-TRIPLES [GB04] and N3 notations [BL98]. For instance, the previous graph can be represented in RDF/XML as:

```
<rdf:RDF xmlns="http://www.eurocup.org#"
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#">

    <rdf:Description rdf:about="http://www.eurocup.org">
        <homePage rdf:parseType="Literal"><html>...</html></homePage>
    </rdf:Description>

    <Team rdf:about="http://www.eurocup.org#Portugal">
        <hasPlayer rdf:parseType="Resource">
            <rdf:type rdf:resource="http://www.eurocup.org#LeftMidfielder"/>
            <rdf:type rdf:resource="http://www.eurocup.org#RightMidfielder"/>
            <dateOfBirth>1972-11-04</dateOfBirth>
            <name xml:lang="pt">Luis Figo</name>
            <number rdf:datatype="http://www.w3.org/2001/XMLSchema#integer">
            7
            </number>
        </hasPlayer>
        <represents>Portugal</represents>
    </Team>

    <rdf:Description rdf:about="http://www.eurocup.org#groupA">
        <hasTeam rdf:resource="http://www.eurocup.org#Greece"/>
        <hasTeam rdf:resource="http://www.eurocup.org#Portugal"/>
        <hasTeam rdf:resource="http://www.eurocup.org#Russia"/>
        <hasTeam rdf:resource="http://www.eurocup.org#Spain"/>
    </rdf:Description>
</rdf:RDF>
```

For our purposes, the most insightful relationship is the mapping of RDF graphs into a variant of first-order logic, sketched in the LBase W3C working group note [GH03]. The Euro Cup graph can be encoded into first-order logic by the theory:

$$\exists_{?x} \left[\begin{array}{ll} & \texttt{euro:homePage}(\texttt{<http://www.eurocup.org>}, \\ & LiteralValueOf('\texttt{<html>...</html>}', \texttt{rdf:XMLLiteral})) \\ \wedge & \texttt{euro:hasTeam}(\texttt{euro:groupA}, \texttt{euro:Greece}) \\ \wedge & \texttt{euro:hasTeam}(\texttt{euro:groupA}, \texttt{euro:Portugal}) \\ \wedge & \texttt{euro:hasTeam}(\texttt{euro:groupA}, \texttt{euro:Russia}) \\ \wedge & \texttt{euro:hasTeam}(\texttt{euro:groupA}, \texttt{euro:Spain}) \\ & \\ \wedge & \texttt{rdf:type}(\texttt{euro:Portugal}, \texttt{euro:Team}) \\ \wedge & \texttt{euro:represents}(\texttt{euro:Portugal}, 'Portugal') \\ & \\ \wedge & \texttt{euro:hasPlayer}(\texttt{euro:Portugal}, ?x) \\ \wedge & \texttt{rdf:type}(?x, \texttt{euro:LeftMidfielder}) \\ \wedge & \texttt{rdf:type}(?x, \texttt{euro:RightMidfielder}) \\ \wedge & \texttt{euro:name}(?x, pair('\text{Luis Figo}', '\text{pt}')) \\ \wedge & \texttt{euro:number}(?x, LiteralValueOf('7', \texttt{xsd:integer})) \\ \wedge & \texttt{euro:dateOfBirth}(?x, '\text{1972-11-04}') \end{array}\right]$$

The qualified name notation is used once more to simplify the presentation. For instance, $\texttt{euro:dateOfBirth}(?x, '\text{1972-11-04}')$ stands for the first-order atom

$$\texttt{<http://www.eurocup.org\#dateOfBirth>}(?x, '\text{1972-11-04}')$$

It is also a practical convention [GH03] used in the encoding into first-order logic to represent a triple `s rdf:type o.` by the unary predicate $o(s)$. Following this convention, we have the following more natural representation for the above knowledge:

$$\exists_{?x} \left[\begin{array}{ll} & \texttt{euro:homePage}(\texttt{<http://www.eurocup.org>}, \\ & LiteralValueOf('\texttt{<html>...</html>}', \texttt{rdf:XMLLiteral})) \\ \wedge & \texttt{euro:hasTeam}(\texttt{euro:groupA}, \texttt{euro:Greece}) \\ \wedge & \texttt{euro:hasTeam}(\texttt{euro:groupA}, \texttt{euro:Portugal}) \\ \wedge & \texttt{euro:hasTeam}(\texttt{euro:groupA}, \texttt{euro:Russia}) \\ \wedge & \texttt{euro:hasTeam}(\texttt{euro:groupA}, \texttt{euro:Spain}) \\ & \\ \wedge & \texttt{euro:Team}(\texttt{euro:Portugal}) \\ \wedge & \texttt{euro:represents}(\texttt{euro:Portugal}, 'Portugal') \\ & \\ \wedge & \texttt{euro:hasPlayer}(\texttt{euro:Portugal}, ?x) \\ \wedge & \texttt{euro:LeftMidfielder}(?x) \\ \wedge & \texttt{euro:RightMidfielder}(?x) \\ \wedge & \texttt{euro:name}(?x, pair('\text{Luis Figo}', '\text{pt}')) \\ \wedge & \texttt{euro:number}(?x, LiteralValueOf('7', \texttt{xsd:integer})) \\ \wedge & \texttt{euro:dateOfBirth}(?x, '\text{1972-11-04}') \end{array}\right]$$

However, this requires extra axioms to be added:

$$\forall_{?s} (\texttt{euro:Team}(?s) \leftrightarrow \texttt{rdf:type}(?s, \texttt{euro:Team})) \wedge$$
$$\forall_{?s} (\texttt{euro:LeftMidfielder}(?s) \leftrightarrow \texttt{rdf:type}(?s, \texttt{euro:LeftMidfielder})) \wedge$$
$$\forall_{?s} (\texttt{euro:RightMidfielder}(?s) \leftrightarrow \texttt{rdf:type}(?s, \texttt{euro:RightMidfielder}))$$

For simplicity, we stick to the pure binary predicate representation in this section. From the first-order logic RDF representation, the limitations of RDF become evident. A RDF graph corresponds to an existentially closed conjunction of binary atomic formulae. It is not possible to express negations or disjunctions of formulae. Thus it is not possible to express implications, and therefore we need extensions to introduce rules in the languages. But before doing that we need to address the several forms of entailment defined by RDF and RDFS semantics, and corresponding conclusions that can be inferred from a RDF graph.

2.1.2 Entailment

The RDF graph specifies knowledge in the Semantic Web, but it is required a notion of entailment in order to determine what can be concluded from the asserted knowledge. The RDF Semantics [Hay03] specifies four distinct forms of entailment, that we will briefly illustrate through examples using the first-order language encoding. The reader is referred to the normative recommendation for the definition of the several entailment forms, which we proceed to discuss.

Simple entailment The notions of entailment depend on appropriate notions of interpretations and satisfaction, which can be found in [Hay03]. The basic form of entailment allows to test if a RDF graph is entailed by other RDF graph. Simple RDF-entailment does not provide any special meaning to the symbols in the language. The queries must conform to the syntax of RDF graphs, i.e. they are existentially closed conjunction of atoms with binary predicates. Suppose the user wants to query the Euro Cup graph to know whether there is a team with a player named "Luis Figo". This query is represented by the following RDF graph, in N-Triples notation, where `_:x` and `_:y` represent blank nodes:

```
_:x <http://www.eurocup.org#hasPlayer> _:y.
_:y <http://www.eurocup.org#name> "Luis Figo".
```

Intuitively, this RDF graph is simply entailed since `_:x` can be substituted by `rdf:Portugal` and `_:y` by the anonymous resource with value `"Luis Figo"` for `euro:name` property. It is also clear that the query corresponds to the formula below, which is a valid logical conclusion from the Euro Cup graph represented in first-order logic:

$$\exists_{?x,?y} \left[\begin{array}{c} \texttt{<http://www.eurocup.org\#hasPlayer>}(?x, ?y) \\ \wedge \\ \texttt{<http://www.eurocup.org\#name>}(?y, \text{'Luis Figo'}) \end{array} \right]$$

In summary, the conclusions obtained from RDF simple entailment correspond to the first-order conclusions obtained from the translated theory into first-order logic. RDF graph simple entailment can be understood as a subgraph-matching problem, which is known to be decidable but a NP-complete problem [Hay03].

RDF entailment A RDF interpretation captures the fundamental notion of `rdf:Property`, that declares the resources of RDF vocabulary which are properties. Roughly, some resource denotes a property if and only if `rdf:Property` is the `rdf:type` of that resource. By using RDF-entailment, we will be able to extract from the Euro Cup knowledge the following additional conclusions, which are not explicitly stated in that RDF graph:

```
euro:homePage    rdf:type rdf:Property .
euro:hasTeam     rdf:type rdf:Property .
euro:represents  rdf:type rdf:Property .
euro:hasPlayer   rdf:type rdf:Property .
euro:name        rdf:type rdf:Property .
euro:number      rdf:type rdf:Property .
euro:dateOfBirth rdf:type rdf:Property .
```

The other difference to simple RDF entailment is the recognition of syntactically correct `rdf:XMLLiterals` in exclusive canonical form, appearing in property values. Returning to our example, RDF-entailment concludes that

```
<http://www.eurocup.org> euro:homePage _:i1 .
 _:i1 rdf:type rdf:XMLLiteral .
```

or in logical notation

$$\exists_{?i} \left[\begin{array}{c} \texttt{euro:homePage}(\texttt{<http://www.eurocup.org>}, ?i) \\ \wedge \\ \texttt{rdf:type}(?i, \texttt{rdf:XMLLiteral}) \end{array} \right]$$

This is a particularly annoying feature of RDF, since it does not allow us to conclude directly that

```
  "<html>...</html>"^^rdf:XMLLiteral rdf:type rdf:XMLLiteral .
```

because literal values are not allowed in subject nodes. This limitation does not occur in the encoding into first-order logic. For RDF graphs with XML Literals not respecting the Exclusive Canonical Form, one does not conclude under RDF-entailment that these literals have `rdf:XMLLiteral` type.

RDFS entailment With RDFS entailment, the full RDF and RDFS vocabularies can be used together, defining a basic ontological language for the Semantic Web. The major new conceptual novelty is the introduction of classes, as a set of resources. With classes one can construct class hierarchies using `rdfs:subClassOf` statements, define domains and ranges of properties (resorting to `rdfs:domain` and `rdfs:range` properties), as well as hierarchies of sub-properties with `rdfs:subPropertyOf`. RDFS also defines some pre-defined classes, namely `rdfs:Class`, `rdfs:Resource`, `rdfs:Literal`, and `rdfs:Datatype`, just to name a few. This allows to deduce a lot of axiomatic triples from any given RDF-graph. In particular, for each node appearing in the object of a `rdf:type` statement one concludes that these nodes are classes. We illustrate this on our initial Euro Cup graph. For instance, for `euro:Team` the following triples are entailed:

```
euro:Team rdf:type rdfs:Class .
euro:Team rdf:type rdfs:Resource .
euro:Team rdfs:subClassOf euro:Team .
euro:Team rdfs:subClassOf rdfs:Resource .
```

A similar set of statements is obtained for `euro:LeftMidfielder` and `euro:RightMidfielder` (and also for the classes in the RDF and RDFS vocabularies). For properties, it is concluded that they are resources and are sub-properties of themselves. For the `euro:hasTeam` one gets:

```
euro:hasTeam rdf:type rdf:Property .
euro:hasTeam rdf:type rdfs:Resource .
euro:hasTeam rdfs:subPropertyOf euro:hasTeam .
```

Notice that the first triple was already concluded with RDF-entailment. For literals, RDFS-entailment entails that they have type `rdfs:Literal` and `rdfs:Resource`. Syntactically correct `rdf:XMLLiteral`s are treated as in RDF-entailment, but now if a RDF graph has an ill-typed XML Literal the graph becomes inconsistent and everything is entailed from it.

Using the RDFS vocabulary we can improve our knowledge, by asserting the domains and ranges of properties as well as subclass relations. Consider the following new assertions regarding the Euro Cup example:

```
euro:hasTeam  rdfs:domain euro:Group .
euro:hasTeam  rdfs:range euro:Team .

euro:hasPlayer  rdfs:domain euro:Team .
euro:hasplayer  rdfs:range euro:Player .


euro:LeftMidfielder  rdfs:subClassOf euro:Midfielder .
euro:RightMidfielder rdfs:subClassOf euro:Midfielder .

euro:Midfielder rdfs:subClassOf euro:Player .
euro:Goalkeeper rdfs:subClassOf euro:Player .
```

The first 4 triples specify the domains and ranges of properties `euro:hasTeam` and `euro:hasPlayer`. Similar statements can be employed to specify all the domains and ranges of the Euro Cup properties. The last 4 statements assert some of the subclass relationships that hold in the football (or soccer) domain. From the initial Euro Cup graph merged with the previous triples, we obtain a lot more. From the specification of domain and range of `euro:hasTeam` we are able to conclude the following statements, which were originally absent from the RDF graph:

```
euro:groupA rdf:type euro:Group .

euro:Greece rdf:type euro:Team .
euro:Russia rdf:type euro:Team .
euro:Spain rdf:type euro:Team .
```

From the player positions hierarchy RDFS-entailment produces the following new relationships (and similar ones for `euro:RightMidfielder` and `euro:Goalkeeper`):

```
euro:LeftMidfielder rdf:type rdfs:Class .
euro:LeftMidfielder rdf:type rdfs:Resource .
euro:LeftMidfielder rdfs:subClassOf euro:LeftMidfielder .
euro:LeftMidfielder rdfs:subClassOf euro:Midfielder .
euro:LeftMidfielder rdfs:subClassOf euro:Player .
euro:LeftMidfielder rdfs:subClassOf rdfs:Resource .

euro:Midfielder rdf:type rdfs:Class .
```

```
euro:Midfielder rdf:type rdfs:Resource .
euro:Midfielder rdfs:subClassOf euro:Midfielder .
euro:Midfielder rdfs:subClassOf euro:Player .
euro:Midfielder rdfs:subClassOf rdfs:Resource .
```

In particular, for the player with name `"Luis Figo"` we get a lot more information from the subclass relations, as shown below:

```
euro:Portugal euro:hasPlayer _:i5 .

_:i5 euro:dateOfBirth "1972-11-04" .
_:i5 euro:dateOfBirth _:i2 .
_:i5 euro:name "Luis Figo"@pt .
_:i5 euro:name _:i3 .
_:i5 euro:number "7"^^xsd:integer .

_:i5 rdf:type euro:LeftMidfielder .
_:i5 rdf:type euro:Midfielder .
_:i5 rdf:type euro:Player .
_:i5 rdf:type euro:RightMidfielder .
_:i5 rdf:type rdfs:Resource .

_:i2 rdf:type rdfs:Literal .
_:i2 rdf:type rdfs:Resource .

_:i3 rdf:type rdfs:Literal .
_:i3 rdf:type rdfs:Resource .
```

Also notice again the way how type information regarding literals is obtained by introducing new blank nodes associated with the corresponding literals (but not for typed literals other than `rdf:XMLLiteral`). To finish the illustration of the RDFS entailment capabilities, we introduce some new triples in the underlying RDF graph, capturing the notions of line-ups, initial players and substitutes for a match. This can be done as follows:

```
euro:matchGreecePortugal euro:initialPlayer _:p1 .
euro:matchGreecePortugal euro:initialPlayer _:g1 .
...
euro:matchGreecePortugal euro:initialPlayer _:p11 .
euro:matchGreecePortugal euro:initialPlayer _:g11 .

euro:matchGreecePortugal euro:substitutePlayer _:p12 .
euro:matchGreecePortugal euro:substitutePlayer _:g12 .
...
euro:matchGreecePortugal euro:substitutePlayer _:p19 .
euro:matchGreecePortugal euro:substitutePlayer _:g19 .

euro:initialPlayer rdfs:subPropertyOf euro:line-up .
euro:substitutePlayer rdfs:subPropertyOf euro:line-up .
```

Each blank node denotes a player listed in the RDF graph, and either it is asserted that he is an initial player of the match or a substitute one. Finally, we express that the properties `euro:initialPlayer` and `euro:substitutePlayer` are sub-properties of `euro:line-up`. Using RDFS-entailment we conclude that:

```
euro:matchGreecePortugal euro:initialPlayer _:i8 .
euro:matchGreecePortugal euro:initialPlayer _:i9 .
...
euro:matchGreecePortugal euro:substitutePlayer _:i10 .
euro:matchGreecePortugal euro:substitutePlayer _:i11 .
...
euro:matchGreecePortugal euro:line-up _:i8 .
euro:matchGreecePortugal euro:line-up _:i9 .
euro:matchGreecePortugal euro:line-up _:i10 .
euro:matchGreecePortugal euro:line-up _:i11 .
```

Notice the renaming of blank nodes in the concluded graph (the names don't matter, if replaced consistently), for instance `_:i8` might represent the blank node `_:g1` in the initial graph. The new triples are the last ones where we conclude which players make part of the line-up, and are obtained because of the sub-property declarations. The RDF semantics recommendation gives a set of sound and complete rules for RDFS-entailment, as well as for the other previous forms of entailment. The rule used to implement inheritance in the sub-property hierarchies is shown below:

	From	Add
rdfs7	aaa `rdfs:subPropertyOf` bbb . uuu aaa yyy .	uuu bbb yyy .

Notice that this rule when translated using the previous encoding into first-order language results in the following axiom scheme:

	From	Add
rdfs7	`rdfs:subPropertyOf`(?aaa,?bbb) . ?aaa(?uuu,?yyy) .	?bbb(?uuu, ?yyy) .

This has an apparent 2nd-order logic flavour, but can be dealt with by using a different encoding into first order logic (FOL), namely by letting the ternary predicate $Triple(s, p, o)$ represent the triple `s p o`.

Datatype entailment The last form of entailment presented in the RDF semantics document is Datatype entailment. In a nutshell, RDFS entailment is extended for supporting other typed literals, with emphasis in several XML Schema Datatypes like decimals, integers, dates, and strings. An essential characteristic of the datatype entailment is the mandatory existence of a canonical value for every lexical representation of the value. Therefore, not every type can be supported by this form of entailment; for the full list of allowed XML Schema datatypes please check the list in [Hay04].

The datatype semantics deals with ill-formed literals by making the RDF graph inconsistent, identically to the treatment of `rdf:XMLLiteral`s by RDFS entailment. It also specifies rules that allow the mutual substitution of literals that denote the same value, even with different datatypes. For our Euro Cup graph we should obtain the following triples from an inference engine supporting `xsd:decimal`, `xsd:integer` and `xsd:string` types.

```
_:i1 euro:dateOfBirth _:i2 .
_:i1 euro:dateOfBirth "1972-11-04" .
_:i1 euro:dateOfBirth "1972-11-04"^^xsd:string .
_:i1 euro:number _:i3 .
_:i1 euro:number "7"^^xsd:integer .
_:i1 euro:number "7.0"^^xsd:decimal .
_:i2 rdf:type rdfs:Literal .
_:i2 rdf:type rdfs:Resource .
_:i2 rdf:type xsd:string .
_:i3 rdf:type rdfs:Literal .
_:i3 rdf:type rdfs:Resource .
_:i3 rdf:type xsd:decimal .
_:i3 rdf:type xsd:integer .
```

Notice that for plain literals without language tags, we can conclude that they are `xsd:string`s and vice-versa. For integers we always obtain that they are also decimals, but not the other way around. Only for decimals with fractional part 0 we can conclude they are integers. Furthermore, we also obtain that `xsd:integer rdfs:subClassOf xsd:decimal .` No complete set of rules is given for RDF datatype entailment.

2.1.3 Encoding into First-Order Logic

The RDF Semantics recommendation [Hay04] defines a proper model-theory for the several forms of entailment described previously. However, for integration with existing rule systems we briefly discuss in this section the so-called Axiomatic Semantics, in particular the first-order logic L_{base} representation, illustrated before. We discuss solely the case of RDF graphs under RDFS entailment. A first convention is to use a binary predicate $p(s, o)$ to represent triple `s p o .` Additionally, a triple with predicate `rdf:type` is encoded into $o(s)$. Further details of the translations are provided by the rules in Table 1.

Table 1: Translation of N-Triples into L_{base} [GH03]

RDF Expression E	L_{base} **expression** $TR[E]$
a plain literal `"sss"`	`'sss'`, with any internal occurrences of `'''` prefixed with `'\'`
a plain literal `"sss"@ttt`	the term `pair('sss','ttt')`
a typed literal `"sss"^^ddd`	the term `LiteralValueOf('sss',TR[ddd])`
an RDF container membership property name of the form `rdf:_nnn`	`rdf-member(nnn)`
any other URI reference `aaa`	`aaa` or `<aaa>`
a blank node a variable	(one distinct variable per blank node)
a triple `aaa rdf:type bbb .`	`TR[bbb](TR[aaa])` and `rdfs:Class(TR[bbb])`
any other triple `aaa bbb ccc .`	`TR[bbb](TR[aaa],TR[ccc])` and `rdf:Property(TR[bbb])`
an RDF graph	The existential closure of the conjunction of the translations of all the triples in the graph.
a set of RDF graphs	The conjunction of the translations of all the graphs.

The use of such translation has been partially illustrated before with the Euro Cup example. The

translation of a graph should be complemented with axiom schemes in order to capture RDFS entailment. Some of these axiom schemes are listed in Table 2. The table does not include schemes for datatype and container membership properties, and for most of the axiomatic triples. The details can be found in [GH03]. Notice that we are still in a first-order setting, where the previous axiom schemes represent an infinite number of first-order formulae, obtained by substitution of free variables. The F-logic framework discussed in Section 7 can be used to encode the previous axiom schemes in a straightforward way. The axiom schemes can be used to formalize the intended meaning of several constructs of the RDFS vocabulary, most of which can be found in Table 3.

If the meta-information is clearly separated from the object data, then the FOL representations of Table 3 can be used as the meaning for the RDFS statements. Inheritance in class and sub-properties hierarchies are guaranteed by first-order logic. This representation is explored further on in Sections 3.3 and 4.2.2.

2.2 Expressing Ontologies in Description Logics

The idea of the Semantic Web is to describe the meaning of web data in a way suitable for automatic reasoning. This means that a descriptive data (*meta-data*) in machine readable form is to be stored on the web and used for reasoning.

The ontology level of the semantic web is to provide techniques for describing domains, shared by various applications. Such a shared conceptualization of a domain is called *ontology*. An ontology describes concepts of the domain, usually by defining an hierarchy of *classes* of objects in the described domain and binary relations, called *properties*. It should also support automatic reasoning about the domain. The RDF Schema language, discussed above, is thus a simple ontology definition language, where the notion of entailment provides a basis for reasoning. As pointed out in Section 2.1, RDF (and RDFS) can be related to FOL, in which case RDFS properties can be seen as binary predicates and the triples play the role of atomic formulae. Similarly, classes can be seen as unary predicates.

RDFS seen as ontology language is not very expressive. For example it does not allow to define new classes from given classes, or from given properties, by set-theoretic operations such as disjunction, intersection, complement, projection, etc. Such description mechanisms are used in Description Logics (DLs). DLs is a family of Knowledge Representation formalisms, they are decidable subsets of FOL. Here we briefly summarize main notions, while the reader is referred to [BCM$^+$03] for further details.

The syntax of a DL is built over the distinct alphabets of *class names* $\mathcal{C}$ (also known as *concepts*), *property names* $\mathcal{R}$ (also known as *roles*) and individual names $\mathcal{O}$. Depending on the kind of DL, different constructors are provided to build class expressions (or briefly *classes*) and property expressions (or briefly *properties*). Intuitively, class expressions are used to represent sets of individuals of a domain and property expressions are used to represent binary relations over individuals. Individual names are used to represent individuals of a domain and can be seen as logical constants. In Description Logics it is often assumed that different names represent different individuals of the domain (*unique name* assumption).

An ontology specifies class inclusion relation between its classes and property inclusion relation between its properties. These can be expressed in Description Logics by so called *terminological axioms* of the form $X \sqsubseteq Y$ where both X and Y are either class expressions or property expressions. An expression of the form $X \equiv Y$ will be called an equality axiom. It can be seen as a shorthand for two axioms $C \sqsubseteq D$ and $C \sqsubseteq D$. A set of terminological axioms is often called a T-box.

Expressions of the form $a : C$ and $\langle a, b\rangle : P$ where C is a class expression, R is a property expression and a, b are individual names will be in the sequel called *DL-atoms*. DL-atoms may be used as axioms called *assertions*, stating class membership or property membership. A set of assertions is

Table 2: Axiom Schemes for RDFS entailment (adapted from [GH03])

```
%RDF axioms

rdf:type(?x,?y) implies ?y(?x)

rdf:Property(rdf:type)
rdf:Property(rdf:subject)
rdf:Property(rdf:predicate)
rdf:Property(rdf:object)
rdf:Property(rdf:first)
rdf:Property(rdf:rest)
rdf:Property(rdf:value)
rdf:List(rdf:nil)

NatNumber(?x) implies rdf:Property(rdf-member(?x))

pair(?x,?y)=pair(?u ?v) iff (?x=?u and ?y=?v)

%RDFS axioms

rdfs:Resource(?x)
rdfs:Class(?y) implies (?y(?x) iff rdf:type(?x,?y))

rdfs:domain(?x,?y) implies (forall(?u ?v)(?x(?u,?v)) implies ?y(?u)))

rdfs:range(?x,?y) implies (forall(?u ?v)(?x(?u,?v)) implies ?y(?v)))

rdfs:subClassOf(?x,?y) implies
  (rdfs:Class(?x) and rdfs:Class(?y) and
   (forall (?u)(?x(?u) implies ?y(?u)))

rdfs:Class(?x) implies
  ( rdfs:subClassOf(?x,?x) and rdfs:subClassOf(?x,rdfs:Resource) )

( rdfs:subClassOf(?x,?y) and rdfs:subClassOf(?y,?z) ) implies
  rdfs:subClassOf(?x,?z)

rdfs:subPropertyOf(?x,?y) implies
  (rdf:Property(?x) and rdf:Property(?y) and
   (forall (?u ?v)(?x(?u,?v) implies ?y(?u,?v)))

rdf:Property(?x) implies rdfs:subPropertyOf(?x,?x)

( rdfs:subPropertyOf(?x,?y) and rdfs:subPropertyOf(?y,?z) ) implies
  rdfs:subPropertyOf(?x,?z)
```

Table 3: Representation of RDFS statements in FOL

RDFS statement	Axiom Instantiation	FOL short representation
`rdf:type(s,C)`	`rdfs:Class(C) implies (C(s) iff rdf:type(s,C)) rdfs:Class(C)`	$C(s)$
`rdfs:domain(P,C)`	`rdfs:domain(P,C) implies (forall(?u ?v)(P(?u,?v)) implies C(?u)))`	$\forall_{?u,?v}\,(P(?u,?v) \rightarrow C(?u))$
`rdfs:range(P,C)`	`rdfs:range(P,D) implies (forall(?u ?v)(P(?u,?v)) implies D(?v)))`	$\forall_{?u,?v}\,(P(?u,?v) \rightarrow D(?v))$
`rdfs:subClassOf(C,D)`	`rdfs:subClassOf(C,D) implies (rdfs:Class(C) and rdfs:Class(D) and (forall(?u)(C(?u) implies D(?u)))`	$\forall_{?u}\,(C(?u) \rightarrow D(?u))$
`rdfs:subPropertyOf(P,Q)`	`rdfs:subPropertyOf(P,Q) implies (rdf:Property(P) and rdf:Property(Q) and (forall(?u ?v)(P(?u,?v) implies Q(?u,?v)))`	$\forall_{?u,?v}\,(P(?u,?v) \rightarrow Q(?u,?v))$

often called A-box.

Class expressions, property expressions and axioms can be seen as an alternative representation of FOL formulae. More precisely, individuals are equivalent to FOL constants, class expressions are equivalent to FOL formulae with one free variable and property expressions to formulae with two free variables. For example, class expression C where C is a class name corresponds to the FOL formula $C(x)$, and property expression R where R is a property name corresponds to the FOL formula $R(x, y)$, where x and y are free variables. This applies also to expressions built with constructors. The inclusion axioms are equivalent to the universally quantified implications, e.g. $R \sqsubseteq S$, where R and S are property names corresponds to the formula $\forall x, yR(x, y) \rightarrow S(x, y)$. The assertions correspond to atomic formulae, e.g. $a : C$ corresponds to $C(a)$. Thus, the semantics of DLs is defined by referring to the usual notions of interpretation and model.

Due to the restricted syntax Description Logics are decidable. There are well developed automatic reasoning techniques. Some Description Logics support in addition reasoning with concrete datatypes, such as strings or integers. In that case one distinguises between individual-valued properties and data-valued properties.

Particular Description Logics are denoted by names indicating the kind of allowed constructors, for more details see [BCM$^+$03].

A finite set of axioms in a Description Logic can be considered an ontology. This idea has been adopted by the Semantic Web community. The Web Ontology Language OWL accepted in February 2004 as a W3C Recommendation [SWM03] comes in three variants: Lite, DL and Full. OWL Lite is a subset of OWL DL, which in turn is a syntactic variant of an expressive Description Logic, $\mathcal{SHOIN}(\mathbf{D})$ supporting concrete datatatypes. As a language based on a DL, OWL DL has a well-defined semantics, and is supported by complete reasoning algorithms. A mapping from its abstract syntax to RDF graphs is defined. OWL DL is layered on a subset of RDFS, enforcing strict separation of classes, properties and individuals. OWL Full includes unrestricted RDFS.

In the rest of this section, as an example of a DL, we give a brief overview of the DL $\mathcal{SHOIN}$, abstracting from the concrete datatypes. We also show how the constructs of $\mathcal{SHOIN}$ are reflected by OWL DL class descriptions and axioms.

As usual, $\mathcal{SHOIN}$ is built over a signature of distinct sets of class ($\mathcal{C}$), property ($\mathcal{R}$) and individual ($\mathcal{O}$) names. The set of all $\mathcal{SHOIN}$ properties is equal to the set of property names $\mathcal{R}$ union the set of the inverse properties $\{R^- \mid P \in \mathcal{R}\}$. As usual, $\mathcal{SHOIN}$ admits inclusion and equality axioms and assertions, both for classes and for properties.

In addition, a special kind of axiom makes it possible to assert that a property is transitive.

A $\mathcal{SHOIN}$ property is said to be *simple*, iff it is neither transitive, nor has any transitive sub-properties, and its inverse is also a simple property.

The set of all $\mathcal{SHOIN}$ classes is the smallest set such that every class name in $\mathcal{C}$ and the symbols $\top$, $\bot$ are classes, and if C,D are classes, i is an individual name from $\mathcal{O}$, R is a property, S is a simple property and n an integer, then $\neg C$, $\{i\}$, $(C \sqcap D)$, $(C \sqcup D)$, $(\forall R.C)$, $(\exists R.C)$, $\geqslant n\, S.$, and $\leqslant n\, S.$ are classes.

The semantics of $\mathcal{SHOIN}$ is given by interpretations, where an *interpretation* $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ consists of a nonempty domain $\Delta^{\mathcal{I}}$ and a interpretation function $\cdot^{\mathcal{I}}$. The interpretation function maps classes into subsets of $\Delta^{\mathcal{I}}$, individual names into elements of $\Delta^{\mathcal{I}}$, and property names into subsets of the cartesian product of $\Delta^{\mathcal{I}}$ ($\Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$). Compound class expressions are interpreted according to the following equations

$$\begin{array}{rr}
\top^{\mathcal{I}} = \Delta^{\mathcal{I}} & (C \sqcap D)^{\mathcal{I}} = C^{\mathcal{I}} \cap D^{\mathcal{I}} \\
\bot^{\mathcal{I}} = \emptyset & (C \sqcup D)^{\mathcal{I}} = C^{\mathcal{I}} \cup D^{\mathcal{I}} \\
\neg C^{\mathcal{I}} = \Delta^{\mathcal{I}} \setminus C^{\mathcal{I}} & \{i\}^{\mathcal{I}} = \{i^{\mathcal{I}}\} \\
(\forall R.C)^{\mathcal{I}} = \{x \in \Delta^{\mathcal{I}} \mid \forall y (x,y) \in R^{\mathcal{I}} \Rightarrow y \in C^{\mathcal{I}}\} & \\
(\exists R.C)^{\mathcal{I}} = \{x \in \Delta^{\mathcal{I}} \mid \exists y (x,y) \in R^{\mathcal{I}} \wedge y \in C^{\mathcal{I}}\} & \\
(\geqslant n\, R)^{\mathcal{I}} = \{x \in \Delta^{\mathcal{I}} \mid \sharp\{y \mid (x,y) \in R^{\mathcal{I}}\} \geq n\} & \\
(\leqslant n\, R)^{\mathcal{I}} = \{x \in \Delta^{\mathcal{I}} \mid \sharp\{y \mid (x,y) \in R^{\mathcal{I}}\} \leq n\} &
\end{array}$$

A property and its inverse must be interpreted according to the equation

$$(R^-)^{\mathcal{I}} = \{(x,y) \in \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}} \mid (y,x) \in R^{\mathcal{I}}\}.$$

In addition, the interpretation function must satisfy the transitivity assertions on property names; i.e., for any R declared as transitive if $(x,y) \in R^{\mathcal{I}}$ and $(y,z) \in R^{\mathcal{I}}$, then $(x,z) \in R^{\mathcal{I}}$.

Axioms $C \sqsubseteq D$, $i : C$, $\langle i_1, i_2 \rangle : R$ and $R \sqsubseteq R'$ are satisfied by an interpretation $\mathcal{I}$ iff respectively $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$, $i^{\mathcal{I}} \in C^{\mathcal{I}}$, $(i_1^{\mathcal{I}}, i_2^{\mathcal{I}}) \in R^{\mathcal{I}}$ and $R^{\mathcal{I}} \sqsubseteq R'^{\mathcal{I}}$; an ontology $\mathcal{K}$ is satisfied by $\mathcal{I}$ iff $\mathcal{I}$ satisfies every axiom in $\mathcal{K}$; $C \sqsubseteq D$, $i : C$ and $\langle i_1, i_2 \rangle : R$ w.r.t. $\mathcal{K}$ iff respectively $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$, $i^{\mathcal{I}} \in C^{\mathcal{I}}$ and $(i_1^{\mathcal{I}}, i_2^{\mathcal{I}}) \in R^{\mathcal{I}}$ in every interpretation $\mathcal{I}$ of $\mathcal{K}$.

Figure 3 shows how OWL DL statements correspond to $\mathcal{SHOIN}$ axioms, where C (possibly subscripted) is a class, P (possibly subscripted) is a property, P^- is the inverse of P, P^+ is the transitive closure of P, i (possibly subscripted) is an individual and $\top$ is an abbreviation for $A \sqcup \neg A$ for some class A (i.e., the most general class, called "Thing" in OWL DL).

It can be seen that OWL DL statements can be reduced to class/property inclusion axioms and ground facts (asserted class-instance and instance-property-instance relationships). In the case of `transitiveProperty`, however, the axiom $P^+ \sqsubseteq P$ is taken to be equivalent to asserting that P is a transitive property ($\mathcal{SHOIN}$ does not include the transitive closure operator).

As in any DL, OWL DL classes can be names (URIs) or *expressions*, and a variety of *constructors* are provided for building class expressions. Figure 4 summarises the available constructors and their correspondence with $\mathcal{SHOIN}$ class expressions.

Statement	DL Syntax	Example
`subClassOf`	$C_1 \sqsubseteq C_2$	Human $\sqsubseteq$ Animal $\sqcap$ Biped
`equivalentClass`	$C_1 \equiv C_2$	Man $\equiv$ Human $\sqcap$ Male
`subPropertyOf`	$P_1 \sqsubseteq P_2$	hasDaughter $\sqsubseteq$ hasChild
`equivalentProperty`	$P_1 \equiv P_2$	cost $\equiv$ price
`disjointWith`	$C_1 \sqsubseteq \neg C_2$	Male $\sqsubseteq \neg$Female
`sameAs`	$\{i_1\} \equiv \{i_2\}$	$\{$President_Bush$\} \equiv \{$G_W_Bush$\}$
`differentFrom`	$\{i_1\} \sqsubseteq \neg\{i_2\}$	$\{$john$\} \sqsubseteq \neg\{$peter$\}$
`inverseOf`	$P_1 \equiv P_2^-$	hasChild $\equiv$ hasParent$^-$
`symmetricProperty`	$P^- \equiv P$	sibling$^-$ $\equiv$ sibling
`transitiveProperty`	$P^+ \sqsubseteq P$	ancestor$^+$ $\sqsubseteq$ ancestor
`functionalProperty`	$\top \sqsubseteq \leqslant 1\, P$	$\top \sqsubseteq \leqslant 1$ hasMother
`inverseFunctionalProperty`	$\top \sqsubseteq \leqslant 1\, P^-$	$\top \sqsubseteq \leqslant 1$ isMotherOf$^-$
`range`	$\top \sqsubseteq \forall P.C$	$\top \sqsubseteq \forall$hasParent.Human
`domain`	$\top \sqsubseteq \forall P^-.C$	$\top \sqsubseteq \forall$hasParent$^-$.Human
i `type` C	$i : C$	john : Man
$i_1\, P\, i_2$	$\langle i_1, i_2 \rangle : P$	$\langle$john, peter$\rangle$: hasParent

Figure 3: OWL DL statements and $\mathcal{SHOIN}$ axioms

Constructor	DL Syntax	Example
`intersectionOf`	$C_1 \sqcap \ldots \sqcap C_n$	Human $\sqcap$ Male
`unionOf`	$C_1 \sqcup \ldots \sqcup C_n$	Doctor $\sqcup$ Lawyer
`complementOf`	$\neg C$	$\neg$Male
`oneOf`	$\{i_1 \ldots i_n\}$	$\{$john, mary$\}$
`someValuesFrom`	$\exists P.C$	$\exists$hasChild.Lawyer
`allValuesFrom`	$\forall P.C$	$\forall$hasChild.Doctor
`hasValue`	$\exists P.\{i\}$	$\exists$citizenOf.$\{$USA$\}$
`minCardinality`	$\geqslant n\, P$	$\geqslant 2$ hasChild
`maxCardinality`	$\leqslant n\, P$	$\leqslant 1$ hasChild
`cardinality`	$= nP$	$= 2$ hasParent

Figure 4: OWL DL class constructors

2.3 Rules for the Semantic Web

The existing proposals for building a rule layer on top of the ontology layer of the Semantic Web refer to rule formalisms originating from the field of Logic Programming. This section outlines basics of these formalisms with the intention to facilitate reading of the rest of this document. For a more detailed introduction see e.g. [BG94, Llo87a]. Logic Programming languages, like Prolog, extend the pure rule formalism with built-in procedures and other features which will not be discussed here.

The Syntax

The rules we consider are built over a first-order vocabulary including disjoint sets of function symbols, predicates and variables. The nullary function symbols are called constants. In contrast to DL, where predicates can only be unary or binary, no restriction is placed on the arity of the predicates. DL does not allow either the function symbols other than constants. The *terms* are built in the usual way. As usual, we consider *atoms* of the form $p(t_1, \ldots, t_n)$, where p is an n-ary predicate symbol and $t_1, \ldots, t_n$ are terms, *classical literals* (or *literals*), which are atoms or negated atoms (atoms preceded by $\neg$) and *negation as failure* literals which are literals preceded by $\sim$.

In the papers which are subject of this survey different notational conventions are used for representing predicates, constants and variables. We follow these conventions to avoid confusion of the reader who may be willing to look for details into the source papers.

An *ordinary* (a.k.a. *normal*[1]) logic program is a set of *rules* each having the form:

$$H \quad \leftarrow \quad B_1 \wedge \ldots \wedge B_m \wedge \sim B_{m+1} \wedge \ldots \wedge \sim B_n$$

where H, B_i are classical literals, $n \geq m \geq 0$ and $\sim Bj$ are called *negation-as-failure* literals (NAF literals).

H is called the *head* (a.k.a. *consequent*) of the rule;
$B_1 \wedge \ldots \wedge B_m \wedge \sim B_{m+1} \wedge \ldots \wedge \sim B_n$
is called the *body* (a.k.a. *antecedent*) of the rule. $\leftarrow$ is to be read as "if", so that the overall rule should be read as "[head] if [body]", i.e., "if [body] then [head]", with universal quantification on the outer level. A body including several literals is to be understood as their conjunction. A NAF literal $\sim B_i$ intuitively means B_i is unknown or false, i.e. it is not known to be true (see the next section for discussion of the semantics of rules). If $n = 0$, then the body is empty, i.e., $True$, and notationally the " $\leftarrow$ " is often omitted. A *fact* is a rule whose body is empty and whose head is a ground atom. A rule is said to be *safe* if the variables of the head appear also in some classical body literals.

Specific classes of programs discussed in the literature include restricted kinds of the above general syntax:

- *Definite programs*, where negated atoms and NAF-literals are not allowed in the rules.
- *Datalog programs*, are logic programs where function symbols other than constants are not allowed. Usually it is also required that the rules are safe.

Several extensions of ordinary rules are discussed in the literature. We mention a few of them:

[1]The original notion of normal logic program see e.g. [Llo87a], does not allow classical literals in the rules. Therefore the logic programs discussed here are often called *extended*, see for example [BG94]. The above definition of a normal program is used, among others, in the papers [GHVD03b, ELST04a], which are subject of our survey.

- Rules admitting conjunction of literals in the head. Such a rule can be considered a shorthand for a finite number of ordinary rules.
- Rules admitting disjunction of literals in the head, give rise to *disjunctive logic programming*.
- Another extension, discussed in Section 3.3 introduces priorities on rules and makes distinction between *strict rules* and *defeasible rules*, in order to resolve conflicting conclusions obtained by rule-based reasoning.

A rule is said to be *ground* iff it does not include variables. A *grounding substitution* for a rule is a mapping from its variables into ground terms. By $ground(P)$ we denote the set of ground rules obtained from a program P by all grounding substitutions.

For exchange of rules on the Web a standard XML encoding for the rules is needed. An effort in that direction was undertaken by the RuleML initiative (www.ruleml.org). The proposal RuleML0.87 provides a natural mark-up for Datalog rules, using XML tags such as `<head>, <body>, <atom>` (see the example in Section 3.2.4).

The semantics

Definite Programs We discuss first the semantics of definite programs, not including negated atoms or NAF literals. Following the intuition mentioned above, a rule of the form, built of atoms,

$$H \leftarrow B_1 \wedge \ldots \wedge B_k$$

can be formalized as universally quantified implication, and is thus (by de Morgan law) equivalent to logic formula of the form

$$H \vee \neg B_1 \vee \ldots \vee \neg B_k$$

called *definite Horn clause*, or *Horn rule*. We will say that this Horn rule *corresponds* to the definite LP rule that has the same syntactic form, and vice versa. Likewise, we say that a Horn ruleset $\mathcal{RH}$ and a definite LP ruleset $\mathcal{RP}$ correspond to each other when their rules do (isomorphically). We then also say that $\mathcal{RP}$ is the *LP-correspondent* of $\mathcal{RH}$, and conversely that $\mathcal{RH}$ is the *Horn-correspondent* of $\mathcal{RP}$.

As mentioned above, it is implicit in this notation that all logical variables are universally quantified at the outer level, i.e., over the scope of the whole clause. E.g., the rule

$uncle(x, y) \leftarrow brother(x, z) \wedge father(z, y)$

can be written equivalently as:

$\forall x, y, z.\ uncle(x, y) \leftarrow brother(x, z) \wedge father(z, y).$

The semantics of definite logic programs can now be given in terms of logic, as the *least Herbrand model* of a given program. This is the set of all ground atomic logical consequences of the program. It is indeed a model of the program in classical sense (for details see e.g. [Llo87a]). The least Herbrand model is the smallest (w.r.t. set inclusion) set $\mathcal{S}$ of ground atoms such that for any rule

$H \leftarrow B_1 \wedge \ldots \wedge B_m$,

if $B_1 \wedge \ldots \wedge B_m \in \mathcal{S}$ then $H \in \mathcal{S}$.

The basic reasoning problem for definite LP is to find for a given atom A, called the atomic query, all its ground instances which are logical consequences of the program. So for example for a definite program consisting of the rule above and of the ground facts $brother(john, bob), father(bob, mary), father(bob, ann)$, all the answers to the atomic query $uncle(john, y)$, where y is a variable are $uncle(john, mary)$ and $uncle(john, ann)$.

One distinguishes between *backward* reasoning, where an atomic query is matched/unified with the head of a rule and replaced by the respective instance of the body, and *forward* reasoning, where the head of a ground instance of a rule is added to the conclusion set when all body atoms of this rule are already included in the conclusion set. Backward reasoning for definite logic programs is based on the *SLD resolution* principle, which is sound and complete w.r.t. the least Herbrand model semantics. In the case of a definite Datalog program the least Herbrand model is finite and can be effectively computed by forward reasoning. This case is particularly interesting for the purpose of integrating rules and DL-based ontologies, since the latter do not involve function symbols other than individual constants. In the sequel, the class of equality-free definite Datalog programs will be denoted *def-LP*. The syntactically corresponding fragment of FOL is definite equality-free Datalog Horn FOL, which we call *def-Horn*. Let $\mathcal{RP}$ be a def-LP. Let $\mathcal{RH}$ stand for the corresponding def-Horn ruleset. The conclusion set of $\mathcal{RP}$ then coincides with the smallest (again, w.r.t. set inclusion) Herbrand model of $\mathcal{RH}$.

Next, we discuss this relationship.

The def-LP and the def-Horn ruleset entail exactly the same set of facts. Every conclusion of the def-LP is also a conclusion of the def-Horn ruleset. Relative to the def-Horn ruleset, the def-LP is thus sound; moreover, it is complete for fact-form conclusions, i.e., for queries whose answers amount to conjunctions of facts. However, the def-LP is a mildly *weaker* version of the def-Horn ruleset, in the following sense. Every conclusion of the def-LP must have the form of a fact. By contrast, the entailments, i.e., conclusions, of the def-Horn ruleset are not restricted to be facts. E.g., suppose $\mathcal{RH}$ consists of the two rules

$kiteDay(Tues) \leftarrow sunny(Tues) \wedge windy(Tues)$

and

$sunny(Tues)$.

Then it entails

$kiteDay(Tues) \leftarrow windy(Tues)$

(a non-unit derived clause) whereas $\mathcal{RP}$ does not. In practical applications, however, quite often only the fact-form conclusions are desired — e.g., an application might be interested above only in whether or not $kiteDay(Tues)$ is entailed. The def-LP has the virtue of conceptual and computational simplicity. To use an analogy, like a hard-boiled detective from a mid-century cop story, it says "give me the facts, ma'am, just the facts". Thinking in terms of expressive classes, we will view def-LP as an *expressive subset* of def-Horn— we will call it the expressive *f-subset*. def-LP is a mild weakening of def-Horn along the dimension of entailment power, permitting only fact-form conclusions — we will call this *f-weakening*.

In return for this f-weakening, def-LP has some quite attractive computational characteristics (as well as being expressively extensible in directions that FOL is not, as discussed earlier). For the propositional case of def-LP, exhaustive inferencing is $O(n)$ where $n = |\mathcal{RP}|$ — i.e., worst-case linear time [DG84]. For the general case with logical variables, the entire conclusion set of a def-LP $\mathcal{RP}$ can be computed in time $O(n^{v+1})$, when there is a constant bound v on the number of logical variables per rule (this restriction, which we will call *VB*, is typically met in practice). Inferencing in def-LP is thus tractable (worst-case polynomial time) given VB. In contrast, DLs are generally not tractable (typically ExpTime or even NExpTime complexity for key inference problems).

Normal Programs Definite programs are a special case of normal programs, so the semantics of normal program should reduce to the least Herbrand model semantics for the special case of definite programs. The main approaches to defining semantics of normal programs include the *well-founded*

semantics [VRS91][2] and the *answer set semantics* (or *stable model semantics*)[GL88, GL91]. Formal presentation of these semantics is outside of the scope of this paper. We only briefly sketch some ideas of the *answer set semantics*. We restrict our attention to the normal programs not including function symbols of arity greater than zero.

Normal programs allow the use of classical negation in the head and in the body of a rule. If the NAF literals do not appear in a normal program, the program can still be understood as a set of formulae of FOL. Such programs are sometimes called *positive* programs (see e.g. [ELST04a]). By a Herbrand base HB_P of a normal program P we mean the set of all ground classical literals built over the predicates and constants of the program. Notice that this set is finite.

A subset I of HB_P is said to be inconsistent if it includes literals A and $\neg A$ where A is a ground atom in HB_P, otherwise it is said to be consistent. A consistent subset of HB_P is called an interpretation of P. Intuitively, the interpretation asserts the truth of its literals. An interpretation I is a model of P iff for every ground instance r of a rule in P, whenever all literals in the body of r are in I then also the head of r is in I. A positive program may not have a model. For example the program consisting of two facts $p(a)$ and $\neg p(a)$ where a is a constant does not have a model. Assume the contrary, i.e. an interpretation I is a model of this program. Since the bodies of both rules are empty, hence all body atoms are in I. Thus both heads also belong to I, hence I is inconsistent and cannot be an interpretation. It can be proved that whenever a positive program has a model it also has the least model w.r.t. set inclusion, which is called its *answer set*.

We now discuss the *answer set* semantics of normal programs. It is based on the notion of answer set of positive programs. Intuitively, a NAF literal $\sim B$, where B is a classical literal, means that B is false or unknown. Thus, $\sim B$ is false in a given interpretation I iff $B \in I$.

Let $ground(P)$ be the set of all ground instances of a program P; the literals of its clauses are elements of HB_P, and let I be its interpretation that includes a classical literal B.

The *Gelfond-Lifschitz* transformation of P w.r.t I, denoted $P(I)$, is the positive program obtained as follows from $ground(P)$. For every rule r

$$H \leftarrow B_1 \wedge \ldots \wedge B_m \wedge \sim B_{m+1} \wedge \ldots \wedge \sim B_n$$

in $ground(P)$

- remove r if $B_j \in I$ for some $j > m$.
- otherwise replace r by
  $H \leftarrow B_1 \wedge \ldots \wedge B_m$ (i.e. remove all NAF literals from r).

Note, that in first case above $\sim B_j$ can be considered false in I, according to the intuition of $\sim$, thus r is (trivially) satisfied in I. In the second case, all NAF literals of the rule are considered true in I so the rule is satisfied in I if its transformed form is satisfied. Formally, an interpretation I is said to be an *answer set* of P iff it is the answer set of $P(I)$. An answer set of a normal program may or may not exist, and need not be unique if it exists. Notice, that in the special case of a definite program the rules do not include classical negation nor NAF literals. Thus for any interpretation I the Gelfond-Lifschitz transformation of a definite program is an identity transformation, and the least Herbrand model of the program is its unique answer set.

Several answer set engines are available and interesting applications of answer set programming are reported in the literature (see e.g. the online ASP tutorial materials of ESSLLI 2001 by Ilkka Niemelä http://www.tcs.hut.fi/ ini/papers/NT-esslli2001-handout.ps.gz

Answer set semantics can be extended to the case of disjunctive logic programs (see e.g. [Prz91]).

[2]This paper considers normal programs not admitting classical negation.

3 Adding rules to RDF

Grigoris Antoniou and Carlos Viegas Damásio

3.1 Motivation

RDF (and RDFS) allows expressing knowledge in the Semantic Web but does not provide ways of extracting new knowledge from the asserted one. Rules are the standard of-the-shelf logical mechanism to achieve this, and have been thoroughly studied in the last 30 years in the knowledge representation and logic programming communities. Returning to our Euro Cup example, the following rules might be of interest to a Semantic Web agent:

R1) If a team represents a country, and someone is a player of that team then this player has citizenship of that country

R2) If someone is a supporter of a team and there is a match of the team then the supporter buys a ticket for that match

R3) If someone buys a ticket for a match in a given place and at a given date then this person books a hotel for that place, checking in the day of the match and checking out in the next day

Using the translation presented in Section 2.1.3, the above three rules might be encoded into first-order logic by the sentences

$$\forall_{?T,?C,?P}\,[(euro : represents(?T, ?C) \wedge euro : hasPlayer(?T, ?P)) \rightarrow euro : citizenOf(?P, ?C)]$$

$$\forall_{?S,?T}\,[(euro : supporter(?S, ?T) \wedge euro : matchTeam(?M, ?T)) \rightarrow euro : buyTicket(?S, ?M)]$$

$$\forall_{?S,?M,?L,?D}\left[\begin{array}{c}(euro : buyTicket(?S, ?M) \wedge euro : locatedIn(?M, ?L) \wedge euro : matchDate(?M, ?D)) \\ \rightarrow \\ (\exists_{?B}\,(travel : bookHotel(?S, ?B) \wedge travel : city(?B, ?L) \wedge travel : checkin(?B, ?D)))\end{array}\right]$$

The first two rules can be formalized as definite Horn clauses, discussed in Section 2.3, with the more user-friendly syntax:

$$euro : citizenOf(?P, ?C) \leftarrow euro : represents(?T, ?C) \wedge euro : hasPlayer(?T, ?P)$$
$$euro : buyTicket(?S, ?M) \leftarrow euro : supporter(?S, ?T) \wedge euro : matchTeam(?S, ?T)$$

The last one brings some problems, since it allows for a complex, existentially quantified formula in the consequent. This cannot be captured by definite Horn clauses[3].

Since RDF does not allow the assertion of negative or disjunctive knowledge, the extension of RDF to rules is straightforward, as can be seen from the previous example. The RDF triples can be seen as atomic formulae and the rules have the form of definite clauses. The theory obtained in this way must be complemented with the axiom schemes or rules of inference presented in Section 2.1. This is the approach that has been followed in most of the existing RDF(S) reasoners. Consequently the declarative semantics of the extended language can be related to the standard minimal model semantics of van Emden and Kowalski [vEK76].

In this section, we briefly survey several existing systems [CWM, Roo, Jen, SDH02, Dam05b, Wie, Mar, Wal] which implement languages extending RDF in a way that allows to express rules as the above. However, the systems to be discussed have different implementation techniques and allow different language extensions, which we will compare. In particular, some of them support more expressive constructs for dealing with negation, and other mechanisms adapted to the Semantic Web environment.

[3] A possible way out is to use Skolemisation to generate new blank nodes.

3.2 Systems Integrating Rules and RDF

3.2.1 CWM and Euler

The CWM system is a forward engine developed in Python specially designed for the Semantic Web [CWM]. CWM supports N-TRIPLES, RDF/XML and Notation 3 (N3) formats, and it is capable of mutual translation between them. Notation 3 [BL98] is the major format for representing RDF stores, and includes extensions for specifying rules. N3 provides abridged syntax to write RDF graphs supporting:

- formulae, with variables and quantifiers
- the association of prefixes with namespaces
- shortcuts to share the subject as well subject and predicate of several statements
- syntactical notation to refer to blank nodes having some property
- lists, translated into equivalent RDF lists

Furthermore, CWM has several built-ins which implement cryptographic functions, mathematical operations, date and time handling, as well as string operations. The most importation built-ins support several logical representation capabilities and inference mechanisms, which we proceed to illustrate. The three rules stated at the beginning of the section can be encoded in CWM as follows:

```
@prefix euro: <http://www.eurocup.org#> .
@prefix travel: <http://www.travelagency.com#> .

### Rule R1:
{ ?T euro:represents ?C. ?T euro:hasPlayer ?P. } => { ?P euro:citizenOf ?C. }.

### Rule R2:
{ ?S euro:supports ?T . ?M euro:matchTeam ?T. } => { ?S euro:buyTicket ?M. }.

### Rule R3:
{ ?S euro:buyTicket ?M.
  ?M euro:locatedIn ?L. ?M euro:matchDate ?D.
} =>
{ ?S travel:bookHotel [travel:city ?L; travel:checkin ?D] }.
```

The antecedents and consequents of rules are sets of triples implicitly conjoined together, and all variables are universally quantified. The reader should particulary notice the way anonymous nodes can be used in the consequents of rules, as in the last rule, representing an implicit existential quantification. This is a distinguished feature of N3 which cannot be captured in definite Horn clauses. Since the CWM machine has support for time-handling, one can have a more correct rule for handling the check-out date of the hotels. The correct booking rule is:

```
{ ?S euro:buyTicket ?M. ?M euro:locatedIn ?L. ?M euro:matchDate ?D .
  ?D time:inSeconds ?t . ?Din time:inSeconds ?t.
  (?t "86400") math:sum ?t1. ?Dout time:inSeconds ?t1
} =>
{ ?S travel:bookHotel [travel:city ?L; travel:checkin ?Din; travel:checkout ?Dout] }.
```

The CWM system also provides built-ins for obtainining rules and triples from the Web, which are not discussed here. Furthermore, it has functions for testing whether a N3 formula is included, is not included, or is a conclusion of a N3 theory, which can be used in the antecedent. These are meta-reasoning mechanisms of CWM. The non-inclusion test can be used to simulate a kind of default

reasoning, in an awkward and limited manner. CWM does not support a full non-monotonic negation like the one of Answer Set Semantics.

RDF schema entailment is not supported natively by CWM, but can be partially implemented resorting to a set of rules written in N3. These rules can be seen as an implementation of the axiom schemes described in Section 2.1.3. For instance, the following rules implement the handling of properties in RDFS and have been adapted from the `rdfs-rules.n3` module of the Euler system [Roo]:

```
{?S ?P ?O} => {?P a rdf:Property}.

{?P a rdf:Property} => {?P rdfs:subPropertyOf ?P}.

{?Q rdfs:subPropertyOf ?R. ?P rdfs:subPropertyOf ?Q}
=> {?P rdfs:subPropertyOf ?R}.

{?P has rdfs:subPropertyOf ?R. ?S ?P ?O} => {?S ?R ?O}.
```

By using the full set of rules in module `rdfs-rules.n3` jointly with a RDF graph, one will get a sound approximation of the full set of consequences obtained by RDFS-entailment[4]. However, CWM apparently does not properly handle ill-formed XML Literals, and therefore it is not a complete engine for RDF and RDFS. This is somewhat expected since the datatyping rules cannot be represented in N3 notation; they must be in-built in the engine.

Euler [Roo] is a backward engine coded in Java and C# which also supports N3 notation, relying on loop-checking techniques to guarantee termination. The Euler system has internal support for several XML Schema datatypes, allowing to detect inconsistent RDFS theories. Furthermore, it provides several modules implementing the different forms of entailment specified by RDF semantics, as well as for subsets of the OWL language. It also can produce a proof of the query, in N3 notation. However, it is well-known that the loop-checking techniques are inefficient, and cannot be extended to general definite logic programming with complex terms, since they are either unsound or incomplete [BAK91].

3.2.2 Jena

The Jena2 system is the most complex, sophisticated and integrated system for performing RDFS entailment. Internally, it resorts to a hybrid forward and backward engine with tabling support. It provides several reasoners, namely:

- Transitive reasoner
- RDFS rule reasoner
- OWL FB reasoner
- Generic rule reasoner

The transitive reasoner provides the basic inferencing rules for determining the class and property hierarchy, in order to improve performance and diminish space consumption. The transitive reasoner is used in the RDFS rule reasoner for performing the transitive closure of properties `rdfs:subClassOf` and `rdfs:subPropertyOf`. The RDFS rule reasoner has three distinct modes that the user can select:

Full: All the rules for RDFS entailment are used, with the exception of the ones which allocate blank nodes. This is the most expensive mode.

[4]Notice that RDFS entailment generates an infinite number of conclusions from an empty graph

Default: In this mode the rules **rdf1**, **rdfs4a**, and **rdfs4b** are not used as well as detection of container membership properties. Rule **rdf1** states that every predicate of a triple is a `rdf:Property` while **rdfs4a** and **rdfs4b** are used to conclude that the subject or object, respectively, of a triple have type `rdfs:Resource`. An implementation of these rules is discussed below.

Simple: According to the authors, this is the most useful mode where the transitive closures of `rdfs:subClassOf` and `rdfs:subPropertyOf` are obtained, as well as domain and range information and inheritance. All axiomatic triples are not included.

The OWL FB reasoner implements a subset of OWL Lite inference and the generic rule reasoner is used to implement all the others. We illustrate here the use of the generic rule reasoner for implementing RDFS entailment, as well as rules R1 and R2 of our Euro Cup example. The Euro Cup rule in backward direction can be encoded as follows:

```
### Rule R1
(?P euro:citizenOf ?C) <- (?T euro:represents ?C), (?T euro:hasPlayer ?P) .
```

Exchanging the rule symbol we encode rules in forward direction.

```
### Rule R2
(?S euro:supports ?T), (?M euro:matchTeam ?T) -> (?S euro:buyTicket ?M ) .
```

The rules written in backward direction are used by the forward engine, but not vice-versa. Backward rules which appear as consequents of forward rules are ignored in the forward mode, but they are used in the hybrid mode, as discussed below. There is no way of declaring at the rule syntactic level the association of prefixes with URIs. This must be hard-coded in Java, which should be done for the `euro:` prefix.

The non-optimized Jena2 forward RDFS rules for implementing sub-property related entailment are listed below:

```
[rdf1and4: (?x ?p ?y) ->
                (?p rdf:type rdf:Property),
                (?x rdf:type rdfs:Resource), (?y rdf:type rdfs:Resource) ]

[rdfs5a: (?a rdfs:subPropertyOf ?b), (?b rdfs:subPropertyOf ?c) ->
                (?a rdfs:subPropertyOf ?c)]

[rdfs5b: (?a rdf:type rdf:Property) -> (?a rdfs:subPropertyOf ?a)]

[rdfs6:  (?a ?p ?b), (?p rdfs:subPropertyOf ?q) -> (?a ?q ?b)]
```

Notice that multiple triples are allowed in the head, implicitly conjoined together, and the programmer can assign names to the rules. The above rules correspond directly to the RDFS definition and are identical to the ones in the CWM system. A major difference to the CWM system is the possibility of using backward reasoning. One could write rule **rdfs6** as follows:

```
(?a ?q ?b) <- (?p rdfs:subPropertyOf ?q), (?a ?p ?b) .
```

The problem with this rule is that it applies to every predicate and every inference must use it. The problem with the forward rule implementation of **rdfs6** previously shown, is that it applies the inheritance rule for every inferred sub-property, even for those irrelevant to the query. The best of both worlds can be obtained by using the hybrid Jena2 reasoner. We can write **rdfs6** in Jena2 as follows:

```
(?p rdfs:subPropertyOf ?q), notEqual(?p,?q) -> [ (?a ?q ?b) <- (?a ?p ?b) ] .
```

First all forward rules are applied to the given data, thus for every `?p` which is a `rdfs:subPropertyOf` of `?q` the backward rule `(?a ?q ?b) <- (?a ?p ?b)` is added to the rules used by the LP backward engine. We obtain in this way the specific backward rules only for the necessary predicates, with significant performance improvement. Notice the similarity of this rule to an axiom scheme of Table 2.

3.2.3 TRIPLE

TRIPLE [SDH02] is a semantic Web engine supporting RDF and a subset of OWL Lite. Its syntax is based on F-Logic (for more details about F-logic see section 7) and supports an important fragment of first-order logic. A triple is represented by a statement of the form `S[P->O]` and sets of statements sharing the same subject can be grouped together using molecules of the form `S[P1->O1;P2->O2;...]`. All variables must be explicitly quantified, either existentially or universally. Arbitrary formulae can be used in the body, while the heads of rules are restricted to be atoms or conjunctions of molecules. An interesting and useful feature of TRIPLE is a possibility of defining *models* collecting sets of related sentences. For example, a model containing the rules for a sound version of RDFS entailment is presented below. It describes thus the semantics of any given collection of sentences, which is to be provided as actual parameter in the form of a TRIPLE model.

```
// namespace declarations
rdf := "http://www.w3.org/1999/02/22-rdf-syntax-ns#".
rdfs := "http://www.w3.org/TR/1999/PR-rdf-schema-19990303#".

// definition of RDF Schema semantics
FORALL Mdl @rdfschema(Mdl) {
  FORALL O,P,V  O[P->V] <- O[P->V]@Mdl.
  FORALL O,P,V  O[P->V] <- EXISTS S   (S[rdfs:subPropertyOf->P] AND O[S->V]).
  FORALL O,P,V  O[rdfs:subClassOf->V] <-
      EXISTS W  (O[rdfs:subClassOf->W] AND W[rdfs:subClassOf->V]).
  FORALL O,P,V  O[rdfs:subPropertyOf->V] <-
      EXISTS W  (O[rdfs:subPropertyOf->W] AND W[rdfs:subPropertyOf->V]).
  FORALL O,T    O[rdf:type->T] <-
      EXISTS S   (S[rdfs:subClassOf->T] AND O[rdf:type->S]).
}
```

The first two rules of the Euro Cup example are represented in the following TRIPLE model:

```
@eurocup {
    euro := "http://www.example.org#".
    FORALL P,C    P[euro:citizenOf->C] <-
        EXISTS T  ( T[euro:represents->C] AND T[euro:hasPlayer->P] ).
    FORALL S,M    S[euro:buyTicket->M] <-
        EXISTS T  (S[euro:supports->T] AND M[euro:matchTeam->T]).
}
```

The model `rdfschema(euro)` can now be queried in TRIPLE, as illustrated b he following example.

```
 FORALL X,Y <-
X[euro:citizenOf->Y]@rdfschema(eurocup).
```

Notice that the query makes explicit use of the model `rdfschema(Mdl)` with `Mdl` instantiated to `eurocup`. The first rule includes in the instantiated model `rdfschema(eurocup)` all the triples derived at model `eurocup`. The second rule implements sub-property inheritance, while the next two rules provide the transitive closure of properties `rdfs:subClassOf` and `rdfs:subPropertyOf`. The last one defines class inheritance. The rdfschema model is not complete with respect to RDFS entailment, in particular is lacking of support for handling `rdf:XMLLiteral` datatype as well as domain and ranges of properties. TRIPLE provides many other features, not discussed here, like path expressions, Skolem model terms, as well as model intersection and difference. Finally, it should be mentioned that the queries and models are compiled into XSB Prolog, which guarantees termination of inference. TRIPLE uses Lloyd-Topor's transformation [Llo87a]) to take care of the first-order connectives in the sentences, and thus internally uses negation as failure to support efficiently the more complex constructs.

3.2.4 SEW

The SEW system [Dam05b] is a native sound and complete XSB-Prolog [SSD94] implementation of a RDFS engine, including some XML Schema datatype support. It is capable of reading RDF graphs serialized in RDF/XML and N-TRIPLES formats. Rulebases can be loaded from Prolog and RuleML files. The engine is implemented as a tabulated meta-interpreter and is significantly faster than CWM and Euler, with paraconsistent strong and weak negations under extensions of well-founded semantics. However, it does not support N3 notation and therefore it is currently syntactically restricted to rules with a single triple in the consequent. It is expected that SEW will incorporate the First-Order-Logic RuleML which has the expressive capabilities necessary to capture full N3 notation.

A nice feature of SEW is the capability of storing separately several RDF graphs and rules. The rule **R1** for the Euro Cup knowledge base can be encoded in RuleML 0.87 as follows:

```
<Imp>
  <head>
    <Atom>
      <opr><Rel href="http://www.eurocup.org#citizenOf"/></opr>
      <Var>player</Var>
      <Var>country</Var>
    </Atom>
  </head>
  <body>
    <And>
      <Atom>
        <opr><Rel href="http://www.eurocup.org#represents"/></opr>
        <Var>team</Var>
        <Var>country</Var>
      </Atom>
      <Atom>
        <opr><Rel href="http://www.eurocup.org#hasPlayer"/></opr>
        <Var>team</Var>
        <Var>player</Var>
      </Atom>
    </And>
  </body>
</Imp>
```

Alternatively, the user can resort to ordinary Prolog syntax for specifying rules. Rule **R2** has the following shape:

```
'<http://www.eurocup.org#buyTicket>'(Match,Supporter) :-
    '<http://www.eurocup.org#supports>'(Supporter,Team),
    '<http://www.eurocup.org#matchTeam>'(Match,Team).
```

A RDF graph can be represented as a set of ground facts with some special syntax to represent Blank Nodes and Literals. Since Prolog can easily be used to construct Semantic Web engines we believe it is important to propose a standard presentation for triples in Prolog. The translation of RDF constructs into the representation used in the SEW implementation are depicted at Table 4, and can be used as an appropriate and user-friendly syntax for all Prolog programmers. The implementation relies on the tabulation support by XSB-Prolog, since it is fundamental to guarantee termination for all proofs, namely computation of transitive closures of `rdfs:subClassOf` and `rdfs:subPropertyOf` properties. An abstract form of the code present in the SEW engine which implements the rules for handling properties in RDFS entailment can be found below:

```
% rdfs_entails( ?Predicate, ?Subject, ?Object)

% Rule rdf1
rdfs_entails( '<http://www.w3.org/1999/02/22-rdf-syntax-ns#type>',
              AAA,
```

Table 4: RDF constructs in Prolog

RDF construct	Prolog term representation
S P O .	triple(S,P,O) or P(S,O)
<URI>	'<URI>'
_:ID	bnode(ID)
"literal"	'@'("literal","')
"literal"@language	'@'("literal",language)
"literal"^^<URI>	'^^'("literal",<URI>)

```
            '<http://www.w3.org/1999/02/22-rdf-syntax-ns#Property>'
          ) :- rdfs_entails( AAA, _, _ ).


% Rule rdfs5
rdfs_entails( '<http://www.w3.org/2000/01/rdf-schema#subPropertyOf>', UUU, XXX ) :-
    rdfs_entails( '<http://www.w3.org/2000/01/rdf-schema#subPropertyOf>', UUU, VVV ),
    rdfs_entails( '<http://www.w3.org/2000/01/rdf-schema#subPropertyOf>', VVV, XXX ).

% Rule rdfs6
rdfs_entails( '<http://www.w3.org/2000/01/rdf-schema#subPropertyOf>', UUU, UUU ) :-
    rdfs_entails( '<http://www.w3.org/1999/02/22-rdf-syntax-ns#type>',
                  UUU, '<http://www.w3.org/1999/02/22-rdf-syntax-ns#Property>' ).

% Rule rdfs7
rdfs_entails( BBB, UUU, YYY ) :-
    rdfs_entails( '<http://www.w3.org/2000/01/rdf-schema#subPropertyOf>', AAA, BBB ),
    uri(AAA), uri(BBB),
    rdfs_entails( AAA, UUU, YYY ).
```

The previous code expresses declaratively the rules presented at RDF semantics documents, and cannot be used in ordinary Prolog systems without tabulation since evaluation will not terminate.

3.2.5 Semantic Web Library of SWI-Prolog

The SWI-Prolog[5] distribution includes an efficient Semantic Web library devised for efficient, large and scalable storage of triples, which can be interfaced with Prolog programs. It follows the hybrid approach, by separating the reasoning with triples from Prolog rules. The programmer must resort to special predicates for knowing which properties are sub-properties of other properties, or which classes are subclasses of other classes, but still cycles cannot occur in the `rdfs:subClassOf` and `rdfs:subPropertyOf` properties. Thus, the RDF and RDFS reasoners are not complete.

A view like rule can be written in top of the RDF storage with a Prolog like syntax. For instance, our rule R2 would be:

```
'http://www.eurocup.org#buyTicket'(Match,Supporter) :-
    rdf_has(Supporter,'http://www.eurocup.org#supports',Team),
    rdf_has(Match,'http://www.eurocup.org#matchTeam',Team).
```

The `rdf_has` predicate is the interface predicate provided by the library in order to query the RDF storage, taking into account non-cyclic `rdfs:subPropertyOf` triples stored. The problem of this approach is that the conclusions are not known by the RDF storage, so any derived new triples of sub-properties cannot be propagated to super-properties. Of course, it is possible to store those triples in the RDF storage with a query like the following:

[5] http://www.swi-prolog.org

```
rdfe_transaction(
    'http://www.eurocup.org#buyTicket'(Match,Supporter),
    rdf_assert(Match,'http://www.eurocup.org#buyTicket',Supporter),
    fail
;   true ) ).
```

However, such a forward-like reasoning mechanism requires storage of derived triples in the database, with useless space consumption. Moreover, the programmer must control explicitly the execution of rules when the database is updated, which can be very complex if recursive rules are used.

3.2.6 MetaLog and IBL

The meaning of RDFS rules can be difficult to grasp because of the syntax overload. In this small section we describe two systems which allow the specification of knowledge bases with a controlled English language, namely Internet Business Logic [Wal] and MetaLog [Mar]. IBL is a proprietary system, but it can be tried in the Web while MetaLog is distributed freely in the W3C site. They both allow a user-friendly way of specifying rules, and translate rules into logic programs. As far as we know, IBL uses internally a stratified logic programming language with second order constructs. MetaLog translates statements directly into Prolog, and evaluates them using SWI-Prolog. For instance, class inheritance of RDFS can be specified in IBL with the following rule:

```
some-subject is related by rdf:type to some-subclass
that-subclass is related by rdfs:subClassOf to some-object
rdf:type can be expanded to some-URI1:name1
rdfs:subClassOf can be expanded to some-URI2:name2
-----------------------------------------------------
that-subject is related by rdf:type to that-object
```

The IBL product handles correctly recursive definitions, without entering into cycles. The MetaLog system has a similar approach to knowledge representation and allows user to express knowledge very naturally:

```
EURO represents "http://www.eurocup.org".

PORTUGAL represents "Portugal" from EURO.
GREECE represents "Greece" from EURO.
TEAM represents "Team" from EURO.
GROUPA represents "groupA" from EURO.
GREECEPORTUGAL represents "matchGreecePortugal" from EURO.

SUPPORTS represents "supports" from EURO.
PLAYED represents "hasTeam" from EURO.
BUY represents "buyTicket" from EURO.

the match GREECEPORTUGAL  is PLAYED by PORTUGAL.

the match GREECEPORTUGAL  is also PLAYED  by GREECE .

there is an author of this paper named CARLOS which SUPPORTS the team PORTUGAL.
GRIGORIS is a co-author of this paper that SUPPORTS the GREECE team.

if FAN is a SUPPORTER of a SQUAD and a MATCH is PLAYED by SQUAD
then that FAN will BUY a ticket to MATCH.

can you tell me whether there is a FAN that will BUY a ticket to a
MATCH?
```

The MetaLog system answers with

Table 5: Summary of characteristics of RDF(S) reasoners

System	Type	Languages Supported	Inference Mechanisms	RDF(S) support	Implementation
CWM	homog.	N3, N-Triples RDF/XML	forward	axioms	Python
Euler	homog.	N3	backward (loop-checking)	axioms	Java and C#
Jena	homog.	RDF/XML N-Triples, N3	mixed (with tabling)	axioms	Java
TRIPLE	homog.	F-logic	backward (tabling)	axioms	XSB-Prolog and Java
SEW	homog.	RuleML, N-Triples, RDF/XML	backward (tabling)	rules	XSB-Prolog
SWI	hybrid	RDF/XML	backward	rules	Prolog

```
comment:-------------------------------------------.
comment: query 1 (of 1).
comment: query: "can you tell me whether there is a FAN that will BUY a ticket to a MATCH ?".
comment: Metalog result 1 (of 1).

GREECEPORTUGAL represents "matchGreecePortugal" from "http://www.eurocup.org".
BUY represents "buyTicket" from "http://www.eurocup.org".

FAN BUY GREECEPORTUGAL.
```

Unfortunately, MetaLog translates the above statements into standard Prolog, which is not capable of handling correctly recursive predicates. Furthermore, there is neither in-built support for RDF and RDFS entailment nor it is explained how can it be done.

3.2.7 Discussion

Rule support in RDF(S) is already a practical reality. The common subset to all systems we have analysed is a logic programming language based on definite Horn clauses, with sound and complete proof procedures. CWM [CWM], Euler [Roo], Jena [Jen], TRIPLE [SDH02], SEW [Dam05b] and IBL [Wal] are homogeneous reasoners, according to the classification in the introductory section. We have also presented the SWI-Prolog Library [Wie] which can be seen as a hybrid reasoner. The systems IBL [Wal] and MetaLog [Mar] provide natural language interfaces to reasoning in the Semantic Web. It should be noticed that CWM, Euler, Jena, and TRIPLE use “modules” containing the axiom schemes for the several forms of entailment defined for RDF graphs, while SEW resorts to hard-coded inference rules of the RDF recommendation. Furthermore, CWM is a forward engine while Euler, Jena, TRIPLE and SEW are backward ones. Jena supports both directions, and even can mix them together. Triple and SEW rely on XSB-Prolog tabling features for guaranteeing termination of evaluation, while Euler uses loop-checking techniques. Jena also has its own implementation of tabling, inspired by the mechanisms of XSB-Prolog. Table 5 summarizes characteristics of the systems discussed in this section.

3.3 Defeasible Reasoning with RDFS Ontologies

3.3.1 Motivation

The formal foundation of all Semantic Web languages that have matured so far to lead to Web standards (RDF, RDFS, OWL) are based on classical predicate logic. While valuable and useful in many, if not most, situations, approaches based on classical logic suffer from a serious drawback: they cannot deal with conflicting information in an appropriate manner. Once an inconsistency arises, every conclusion can be drawn from the knowledge base. There are many scenarios in which conflicting rules may arise on the Web. Here we mention a few of them.

- *Reasoning with Incomplete Information*: [AA02] describes a scenario where business rules have to deal with incomplete information: in the absence of certain information some assumptions have to be made which lead to conclusions not supported by classical predicate logic. In many applications on the Web such assumptions must be made because other players may not be able (e.g. due to communication problems) or willing (e.g. because of privacy or security concerns) to provide information. This is the classical case for the use of nonmonotonic knowledge representation and reasoning [MT93].
- *Rules with Exceptions*: Rules with exceptions are a natural representation for policies and business rules [ABM99]. And priority information is often implicitly or explicitly available to resolve conflicts among rules. Potential applications include security policies [APM$^+$04, LGF03], business rules [AA02], personalization, brokering, bargaining, and automated agent negotiations [GDtHO01].
- *Default Inheritance in Ontologies*: Default inheritance is a well-known feature of certain knowledge representation formalisms. Thus it may play a role in ontology languages, which currently do not support this feature. [GGF02] presents some ideas for possible uses of default inheritance in ontologies. A natural way of representing default inheritance is rules with exceptions, plus priority information. Thus, nonmonotonic rule systems can be utilized in ontology languages.
- *Ontology Merging*: When ontologies from different authors and/or sources are merged, contradictions arise naturally. Predicate logic based formalisms, including all current Semantic Web languages, cannot cope with inconsistencies. If rule-based, or Horn definable, ontology languages are used and if rules are interpreted as defeasible (that is, they may be prevented from being applied even if they can fire) then we arrive at nonmonotonic rule systems. A sceptical approach, as adopted by defeasible reasoning, is sensible because does not allow for contradictory conclusions to be drawn. Moreover, priorities may be used to resolve some conflicts among rules, based on knowledge about the reliability of sources or on user input. Thus, nonmonotonic rule systems can support ontology integration.

The basic idea of the work described in this section is to allow for the use of conflicting rules in the presence of ontological knowledge expressed in RDFS. In fact, the logical framework extends naturally to ontologies which lie within the Horn expressible part of OWL, since these ontologies can be given a semantics based on a representation using monotonic rules.

3.3.2 Informal Introduction

Syntactically the rules used in this approach are similar to the ordinary rules of Section 2.3. Their heads are classical literals and bodies are conjunctions of literals and/or NAF literals. the rule body is a conjunction of literals L, or of literals negated

There are two kinds of rules: *Strict rules* are denoted by $A \rightarrow p$, and are interpreted in the classical sense: whenever the premises are indisputable then so is the conclusion. An example of a strict rule is "Professors are faculty members". Written formally:

$$professor(X) \rightarrow faculty(X).$$

Inference from strict rules only is called *definite inference*. Strict rules are intended to define relationships that are definitional in nature, for example ontological knowledge.

Defeasible rules are denoted by $A \Rightarrow p$, and can be defeated by contrary evidence. An example of such a rule is

$$faculty(X) \Rightarrow tenured(X)$$

which reads as follows: "Professors are typically tenured".

A *superiority relation* on R is an acyclic relation $>$ on R (that is, the transitive closure of $>$ is irreflexive). When $r_1 > r_2$, then r_1 is called *superior* to r_2, and r_2 inferior to r_1. This expresses that r_1 may override r_2. For example, given the defeasible rules

$$r : professor(X) \Rightarrow tenured(X)$$
$$r' : visiting(X) \Rightarrow \neg tenured(X)$$

which contradict one another: no conclusive decision can be made about whether a visiting professor is tenured. But if we introduce a superiority relation $>$ with $r' > r$, then we can indeed conclude that visiting professors are not tenured.

The superiority information is used to resolve conflicts among defeasible rules. Note that conflicts may arise, contrary to Horn logic rules, because we have admitted classical negation in the rule heads.

RDFS ontologies can be integrated into this approach through their natural translation into rules capturing their semantics.

$rdf : type(x, C)$	$C(x)$
$rdfs : subClassOf(C, D)$	$C(x) \rightarrow D(x)$
$rdfs : subPropertyOf(P, Q)$	$P(x, y) \rightarrow Q(x, y)$
$rdfs : domain(P, C)$	$P(x, y) \rightarrow C(x)$
$rdfs : range(P, C)$	$P(x, y) \rightarrow C(y)$

Similar transformations can be used for the Horn definable part of OWL, though this feature is currently not supported by any implementation we know of.

3.3.3 An Example

In this section we present a full example of using DR-DEVICE rules in a brokered trade application that takes place via an independent third party, the broker. The broker matches the buyer's requirements and the seller's capabilities, and proposes a transaction when both parties can be satisfied by the trade. In our case, the concrete application is apartment renting and the landlord takes the role of the abstract seller.

Figure 5 describes a potential renter's requirements. These are translated into a logical rules language using defeasible rules and priorities, as shown in Figure 6.

The available apartments are represented in RDF, based on an appropriate RDF Schema ontology. For space limitations, we do not get into more detail; see [BAV04] for more details. The overall approach allows the rules to be applied to the RDFS ontology and the RDF data, and selects an apartment based on the hard requirements and the preferences.

1. Carlos is looking for an apartment of at least 45 sqm with at least 2 bedrooms. If it is on the 3rd floor or higher, the house must have an elevator. Also, pet animals must be allowed.
2. Carlos is willing to pay $300 for a centrally located 45 sqm apartment, and $250 for a similar flat in the suburbs. In addition, he is willing to pay an extra $5 per sqm for a larger apartment, and $2 per sqm for a garden.
3. He is unable to pay more than $400 in total. If given the choice, he would go for the cheapest option. His 2nd priority is the presence of a garden; his lowest priority is additional space.

Figure 5: Verbal description of Carlo's (a potential renter) requirements

$r_1 : \Rightarrow acceptable(X)$
$r_2 : bedrooms(X,Y), Y < 2 \Rightarrow \neg acceptable(X)$
$r_3 : size(X,Y), Y < 45 \Rightarrow \neg acceptable(X)$
$r_4 : \neg pets(X) \Rightarrow \neg acceptable(X)$
$r_5 : floor(X,Y), Y > 2, \neg lift(X) \Rightarrow \neg acceptable(X)$
$r_6 : price(X,Y), Y > 400 \Rightarrow \neg acceptable(X)$
$r_7 : size(X,Y), Y \geq 45, garden(X,Z), central(X) \Rightarrow offer(X, 300 + 2Z + 5(Y - 45))$
$r_8 : size(X,Y), Y \geq 45, garden(X,Z), \neg central(X) \Rightarrow offer(X, 250 + 2Z + 5(Y - 45))$
$r_9 : offer(X,Y), price(X,Z), Y < Z \Rightarrow \neg acceptable(X)$
$r_2 > r_1,\ r_3 > r_1,\ r_4 > r_1,\ r_5 > r_1,\ r_6 > r_1,\ r_9 > r_1$

Figure 6: Declarative description of Carlo's requirements

3.3.4 Classification

The overall approach is homogeneous, according to the classification of Section 1. The ontologies are considered essentially to be sets of rules, based on obvious transformation of RDF, RDF Schema and a subset of OWL into rules, which are then processed as the remaining knowledge expressed in rules.

The approach allows for the use of monotonic and nonmonotonic rules, and preferences among them. Ontologies are written in RDF Schema, or in the Horn- definable part of OWL (though the latter has not yet been implemented).

3.3.5 Semantics and Reasoning

No formal semantics has been explicitly provided yet for the overall logic. However, the semantics can be provided in a straightforward way by combining the RDF model theory [Hay03] with Maher's model semantics for defeasible logic [Mah02].

3.3.6 Implementations

Currently there exist three implementations of the ideas presented above.

DR-Prolog [ABW04] is a system that implements the entire framework described above, and is thus able to reason with: monotonic and nonmonotonic rules, preferences among rules, RDF data and RDFS ontologies; work is underway to allow for the processing of Horn-definable OWL ontologies. The system is implemented by transforming information into Prolog.

DR-DEVICE [BAV04] is another effort on implementing defeasible reasoning. It is implemented in Jess, and integrates well with RuleML and RDF/S. It is a system for query answering. Compared to DR-Prolog, DR-DEVICE exhibits similar functionality, albeit following a different overall approach.

On one hand, the use of Jess requires the translation of logical rules in a non-logical language, with an associated loss in declarativity of the overall approach. On the other hand, it has advantages in that it can potentially integrate more easily with mainstream IT technologies.

SweetJess [GGF02] is another implementation of a defeasible reasoning system (situated courteous logic programs) based on Jess. It integrates well with RuleML. Also, it allows for procedural attachments, a feature not supported by any of the above implementations.However, SweetJess is more limited in flexibility, in that it implements only one reasoning variant. Moreover, it imposes a number of restrictions on the programs it can map on Jess.

In comparison, DR-Prolog and DR-DEVICE implement the full version of defeasible logic. In addition, DR-Prolog has a firm formal foundation provided by a number of papers published in top artificial intelligence and logic programming conferences and journals [ABGM01, AMB00, ABGM00, Mah02, Mah01, MG99, MRA$^+$01]. These works range from formal properties and formal semantics to correctness proofs for transformations used. This theoretical underpinning cannot be claimed by SweetJess and similar approaches.

4 Description Logic Programs

Benjamin Grosof and Ian Horrocks

This section surveys the approach to integration of rules and ontologies proposed in [GHVD03a] and studies the relationship between Description Logics (DLs) and Rules, two Knowledge Representation (KR) paradigms widely used in ontology engineering, and try to characterise the expressive power that is common to both paradigms.

In particular, we will study the intersection of the DL that closely corresponds to web ontology languages such as DAML+OIL and OWL[6] (i.e., $\mathcal{SHIQ}$), and the fragment of First Order Logic (FOL) that corresponds to Declarative Logic Programs (i.e., Horn clauses).

The intersection of DLs and Rules is of interest for several reasons. Firstly, such an intersection could form the common basis for and promote interoperability between DL and Rules based ontology languages; secondly, understanding the expressive *intersection* of these two KRs may help us to understand the expressive *combination/union* of the two KRs; and thirdly, ontologies that are within this intersection may be able to use rules engines to provide scalable reasoning services. Combining Rules with DLs is of particular interest in the context of the Semantic Web and e-Science, where a need has already been identified for expressive power beyond what is provided by DL based ontology languages such as DAML+OIL and OWL, and where efforts are already underway to extend the OWL standard with Horn-like rules [HPS04, HPSB$^+$04].

In the following section we give a more detailed motivation and a technical overview (see Section 2 for details of the syntax and semantics of the various languages that will be discussed). In Section 4.2 we describe in detail a mapping from DL to a fragment of Horn clause logic, and use this mapping to define *Description Logic Programs* (DLP), a Declarative Logic Programming (LP) based language that corresponds closely to the resulting Horn fragment. In Section 4.3 we show how an LP reasoner could be used to perform reasoning tasks, such as class subsumption, that are often of interest in an ontology. Finally, in Section 4.4 we conclude with a discussion of the work that has been presented.

[6] The W3C OWL recommendation actually consists of three languages of increasing expressive power: OWL Lite, OWL DL and OWL Full. *OWL Lite* and *OWL DL* basically very expressive Description Logics (DLs); *OWL Full* provides the same set of constructors as OWL DL, but allows them to be used in an unconstrained way (in the style of RDF), and is therefore (syntactically at least) outside FOL. For this reason we will focus on OWL DL, and we will use OWL in this Section to mean OWL DL.

4.1 Motivation and Overview

4.1.1 Web Services as a Motivation for DLP

A task-oriented motivation for combining LP rules with OWL/DAML+OIL ontologies arises from the efforts to design and build *Semantic Web Services (SWS)*. Semantic Web Services are Web Services that make use of Semantic Web techniques to describe (or implement) services in a knowledge-based manner. The knowledge-based service descriptions may be used for a variety of purposes, including: discovery and search; selection, evaluation, negotiation, and contracting; composition and planning; execution; and monitoring. Efforts to develop Semantic Web Services techniques and to explore their application scenarios include: the DAML-Services effort (DAML-S)[7], the Web Service Modelling Framework effort (WSMF)[8], SweetDeal e-contracting [GLC99, RWG02, GP02], and ECOIN financial knowledge integration [FMG02, FMGar]. Both the DAML-S and SweetDeal efforts have specifically identified combining rules with ontologies as an important requirement. DAML-S began with DAML+OIL (later migrating to OWL) as its main tool for describing services. DAML-S then identified LP rules as desirable in addition. Interestingly, DAML-S has identified LP rules as desirable even to specify ontologies, partly because LP rules are more familiar to mainstream software engineers than DL is.

4.1.2 Overview of the DLP Approach

We start with the goal of understanding the relationship between the two logic based KR formalisms (so as to be able to combine knowledge taken from both): Description Logics (decidable fragments of FOL closely related to propositional modal and dynamic logics [Sch91, Bor96, BCM$^+$03]), and Logic Programs (see, e.g., [BG94] for review) which in turn is closely related to the Horn fragment of FOL. We further focus on def-Horn (a large fragment of Horn FOL), and then go on to show how both DL and LP are related to def-Horn. Highly efficient LP reasoning engines can be used to provide reasoning services for def-Horn.

Our approach is driven by the insight that understanding the expressive *intersection* of these two KRs will be crucial to understanding the expressive *combination/union* of the two KRs. We define a new intermediate KR called *Description Horn Logic (DHL)*, which is contained within this intersection (and so is also a fragment of FOL), and the closely related *Description Logic Programs (DLP)*, which can be viewed as DHL with a moderate weakening of what kind of conclusions can be drawn.

DL and Horn are strict (decidable) subsets of FOL. LP, on the other hand, intersects with FOL but neither includes nor is fully included by FOL. FOL can express (positive) disjunctions, which are inexpressible in LP. There are, however, expressive features of LP, frequently used in practical rule-based applications, that are inexpressible in FOL. One is negation-as-failure, a basic kind of logical non-monotonicity. Another is procedural attachments, e.g., the association of action-performing procedural invocations with the drawing of conclusions about particular predicates.

Description Logic Programs, our newly defined intermediate KR, is contained within the intersection of DL and LP. "Full" LP, including non-monotonicity and procedural attachments, can thus be viewed as including an "ontology sub-language", namely the DLP subset of DL.

Rather than working from the intersection as we do here, one may instead directly address the expressive union of DL and LP by studying the expressive union of DL and Horn within the overall framework of FOL. This is certainly an interesting thing to do. However, to our knowledge, this has not yet been well characterised theoretically, e.g., it is not completely clear how such a union differs from full FOL (see [HPS04]).

[7] `http://www.daml.org/services`
[8] `http://informatik.uibk.ac.at/users/c70385/wese/index.html`

Full FOL has some significant practical and expressive drawbacks as a KR in which to combine DL and rules. First, full FOL has severe computational complexity: it is undecidable in the general case, and intractable even under the Datalog restriction (but see [TRBH04] for work on using an FOL reasoner to reason with OWL ontologies). Second, it is not understood even at a basic research level how to expressively extend full FOL to provide non-monotonicity and procedural attachments; yet these are crucial expressive features in many (perhaps most) practical usages of rules. Third, full FOL and its inferencing techniques are unfamiliar to the great majority of mainstream software engineers, whereas rules (e.g., in the form of SQL-type queries, or Prolog) are familiar conceptually to many of them. The approach we take here avoids these drawbacks by avoiding directly tackling the union (of DL and Horn) in FOL.

DLP provides a significant degree of expressiveness. It is a large fragment of the intersection of DL and LP/Horn, and includes the RDF-Schema (RDFS) [BG00] fragment of DL.

The RDFS fragment of DL permits: stating that a class D is a *Subclass* of a class E; stating that the *Domain* of a property P is a class C; stating that the *Range* of a property P is a class C; stating that a property P is a *Subproperty* of a property Q; stating that an individual b is an *Instance* of a class C; and stating that a pair of individuals (a,b) is an *Instance* of a property P.

Additional DLP expressively permits (within DL): using the *Intersection* connective (conjunction) within *class* descriptions (i.e., in C, D, or E above); using the *Union* connective (disjunction) within *subclass* descriptions (i.e., in D above); using (a restricted form of) *Universal* quantification within *superclass* descriptions (i.e., in E above); using (a restricted form of) *Existential* quantification within *subclass* descriptions (i.e., in D above); stating that a property P is *Transitive*; stating that a property P is *Symmetric*; and stating that a property P is the *Inverse* of a property Q. In RDFS, in contrast, the classes (i.e., C, D, E above) are atomic primitives—they may not have connectives or quantifiers appearing within them.

Via the DLP KR, we give a new technique to combine DL and LP. We show how to perform *DLP-fusion*: the bidirectional mapping of premises and inferences (including typical kinds of queries) from the DLP fragment of DL to LP, and vice versa from the DLP fragment of LP to DL. We call it "DLP-fusion" because it fuses the two logical KRs—DL and LP—-so that information from each can be used in the other. The DLP-fusion technique promises several benefits. We say "promises" because what we present is mainly a theoretical basis; development of detailed algorithms and implementations remain for future work.

In particular, DLP-fusion enables one to "build rules on top of ontologies": it enables the rule KR to have access to DL ontological definitions for vocabulary primitives (e.g., predicates and individual constants) used by the rules. Conversely, the technique enables one to "build ontologies on top of rules": it enables ontological definitions to be supplemented by rules, or imported into DL from rules. It also enables efficient LP inferencing algorithms/implementations, e.g., rule or relational DBMS[9] engines, to be exploited for reasoning over large-scale DL ontologies.

4.2 Mapping DL to def-Horn

In this section we will discuss how DL languages (e.g., DAML+OIL and OWL) can be mapped to def-Horn, and vice versa.

[9]Data Base Management Systems, e.g., SQL query answering systems

4.2.1 Expressive Restrictions

We will first discuss the expressive restrictions of DL and def-Horn as these will constrain the subset of DL and def-Horn for which a complete mapping can be defined.

DLs are decidable subsets of FOL where the decidability is due in large part to their having (a form of) the tree model property [Var97].[10] This property says that a DL class C has a model (an interpretation $\mathcal{I}$ in which $C^{\mathcal{I}}$ is non-empty) iff C has a tree-shaped model, i.e., one in which the interpretation of properties defines a tree shaped directed graph.

This requirement severely restricts the way variables and quantifiers can be used. In particular, quantifiers must be *relativised* via atomic formulae (as in the guarded fragment of FOL [Grä99]), i.e., the quantified variable must occur in a property predicate along with the free variable (recall that DL classes correspond to formulae with one free variable). For example, the DL class $\exists P.C$ corresponds to the FOL formula $\exists y.(P(x,y) \wedge C(y))$, where the property predicate P acts as a guard. One obvious consequence of this restriction is that it is impossible to describe classes whose instances are related to another anonymous individual via different property paths. For example, it is impossible to assert that individuals who live and work at the same location are "HomeWorkers". This is easy with a Horn rule, e.g.:

$$\text{HomeWorker}(x) \quad \leftarrow \quad \text{work}(x,y) \wedge \text{live}(x,z) \wedge \text{loc}(y,w) \wedge \text{loc}(z,w)$$

Another restriction in DLs is that only unary and binary predicates can usually be captured.[11] This is a less onerous restriction, however, as techniques for reifying higher arity predicates are well known [HSTT00].

Definite Horn FOL requires that all variables are universally quantified (at the outer level of the rule), and restricts logical connectives in certain ways. One obvious consequence of the restriction on quantifiers is that it is impossible to assert the existence of individuals whose identity might not be known. For example, it is impossible to assert that all persons have a father (known or unknown). This is easy with a DL axiom, e.g.:

$$\text{Person} \sqsubseteq \exists \text{father}.\top.$$

No negation may appear within the body of a rule, nor within the head. No existentials may appear within the head. Thus it is impossible to assert, e.g., that all persons are either men or women (but not both). This would also be easy using DL axioms, e.g.:

$$\begin{aligned} \text{Person} &\sqsubseteq \text{Man} \sqcup \text{Woman} \\ \text{Man} &\sqsubseteq \neg\text{Woman}. \end{aligned}$$

The Datalog restriction of def-Horn is not an issue for mapping DL into it, since DL also has the Datalog restriction. Finally, the equality-free restriction of def-Horn is a significant restriction in that it prevents representing (partial-)functionality of a property and also prevents representing maximum cardinality. The prohibition against existentials in the head prevents representing minimum cardinality.

4.2.2 Mapping Statements

In this section, we show how (some of) the *statements* (axioms) of DL and DL based languages (such as DAML+OIL and OWL) correspond to def-Horn statements (rules).

[10] Expressive features such as transitive properties and the `oneOf` constructor compromise the tree model property to some extent, e.g., transitive properties can cause "short-cuts" down branches of the tree.

[11] This is not an inherent restriction, and n-ary DLs are known, e.g., $\mathcal{DLR}$ [CDGL98].

RDFS Statements

RDFS provides a subset of the DL statements described in Section 2: subclass, subproperty, range, and domain statements (which in a DL setting are often called Tbox axioms); and asserted class-instance (type) and instance-property-instance relationships (which in a DL setting are often called Abox axioms).

A DL inclusion axiom corresponds to an FOL implication. This leads to a straightforward mapping from class and property inclusion axioms to def-Horn rules as follows:

$C \sqsubseteq D$, i.e., class C is subclass of class D, maps to:
$D(x) \leftarrow C(x)$

$Q \sqsubseteq P$, i.e., Q is a subproperty of P, maps to:
$P(x, y) \leftarrow Q(x, y)$

RDFS range and domain statements correspond to DL axioms of the form $\top \sqsubseteq \forall P.C$ (range of P is C) and $\top \sqsubseteq \forall P^{-}.C$ (domain of P is C). These are equivalent to the FOL sentences $\forall x.\ true \rightarrow (\forall y.\ P(x, y) \rightarrow C(y))$ and $\forall x.\ true \rightarrow (\forall y.\ P(y, x) \rightarrow C(y))$, which can be simplified to $\forall x, y.\ P(x, y) \rightarrow C(y)$ and $\forall x, y.\ P(y, x) \rightarrow C(y)$ respectively. These FOL sentences are already in def-Horn form, which gives us the following mappings for range and domain:

$\top \sqsubseteq \forall P.C$, i.e., the range of property P is class C, maps to:
$C(y) \leftarrow P(x, y)$

$\top \sqsubseteq \forall P^{-}.C$, i.e., the domain of property P is class C, maps to:
$C(y) \leftarrow P(y, x)$

Finally, asserted class-instance (type) and instance-property-instance relationships, which correspond to DL axioms of the form $a : C$ and $\langle a, b \rangle : P$ respectively (Abox axioms), are equivalent to FOL sentences of the form $C(a)$ and $P(a, b)$, where a and b are constants. These are already in def-Horn form: they are simply rules with empty bodies (which are normally omitted):

$a : C$, i.e., the individual a is an instance of the class C, maps to:
$C(a)$

$\langle a, b \rangle : P$, i.e., the individual a is related to the individual b via the property P, maps to:
$P(a, b)$

Note that in these rules a and b are ground (constants).

OWL statements

OWL extends RDF with additional statements about classes and properties (Tbox axioms). In particular, it adds explicit statements about class, property and individual equality and inequality, as well as statements asserting property inverses, transitivity, functionality (unique) and inverse functionality (unambiguous).

DL class and property equivalence axioms can be replaced with a symmetrical pair of inclusion axioms, so they can be mapped to a symmetrical pair of def-Horn rules as follows:

$C \equiv D$, i.e., the class C is equivalent to (has the same extension as) the class D, maps to:
$D(x) \leftarrow C(x)$
$C(x) \leftarrow D(x)$

$P \equiv Q$, i.e., the property P is equivalent to (has the same extension as) the property Q, maps to:

$$\begin{aligned} Q(x,y) &\leftarrow P(x,y) \\ P(x,y) &\leftarrow Q(x,y) \end{aligned}$$

The semantics of inverse axioms of the form $P \equiv Q^-$ are captured by FOL sentences of the form $\forall x, y.P(x,y) \iff Q(x,y)$, and the semantics of transitivity axioms of the form $P^+ \sqsubseteq P$ are captured by FOL sentences of the form $\forall x, y, z.P(x,y) \wedge P(y,z) \rightarrow P(x,z)$. This leads to a direct mapping into def-Horn as follows:

$P \equiv Q^-$, i.e., the property Q is the inverse of the property P, maps to:

$$\begin{aligned} Q(y,x) &\leftarrow P(x,y) \\ P(x,y) &\leftarrow Q(y,x) \end{aligned}$$

$P^+ \sqsubseteq P$, i.e., the property P is transitive, maps to:

$$P(x,z) \leftarrow P(x,y) \wedge P(y,z)$$

DL axioms asserting the functionality of properties correspond to FOL sentences with equality. E.g., a DL axiom $\top \sqsubseteq \leqslant 1\, P$ (P is a functional property) corresponds to the FOL sentence $\forall x, y, z.P(x,y) \wedge P(x,z) \rightarrow y = z$.[12] This kind of axiom cannot be dealt with in our current framework (see Section 4.2.1) as it would require def-Horn rules with equality in the head, i.e., rules of the form $(y = z) \leftarrow P(x,y) \wedge P(x,z)$.

4.2.3 Mapping Constructors

In the previous section we showed how DL axioms correspond with def-Horn rules, and how these can be used to make statements about classes and properties. In DLs, the classes appearing in such axioms need not be atomic, but can be complex compound expressions built up from atomic classes and properties using a variety of constructors. A great deal of the power of DLs derives from this feature, and in particular from the set of constructors provided.[13] In the following section we will show how these DL expressions correspond to expressions in the body of def-Horn rules.

In the following we will, as usual, use C, D to denote classes, P, Q to denote properties and n to denote an integer.

Conjunction (DL $\sqcap$)

A DL class can be formed by conjoining existing classes, e.g., $C \sqcap D$. This corresponds to a conjunction of unary predicates. Conjunction can be directly expressed in the body of a def-Horn rule. E.g., when a conjunction occurs on the l.h.s. of a subclass axiom, it simply becomes conjunction in the body of the corresponding rule

$$C_1 \sqcap C_2 \sqsubseteq D \quad \equiv \quad D(x) \leftarrow C_1(x) \wedge C_2(x)$$

Similarly, when a conjunction occurs on the r.h.s. of a subclass axiom, it becomes conjunction in the head of the corresponding rule:

$$C \sqsubseteq D_1 \sqcap D_2 \quad \equiv \quad D_1(x) \wedge D_2(x) \leftarrow C(x),$$

[12] Note that, technically, this is partial-functionality as for any given x there is no requirement that there exists a y such that $P(x,y)$.

[13] Note that this feature is not supported in the RDFS subset of DLs.

This is then easily transformed into a pair of def-Horn rules:

$$\begin{array}{lcl} D_1(x) & \leftarrow & C(x) \\ D_2(x) & \leftarrow & C(x) \end{array}$$

Disjunction (DL $\sqcup$)

A DL class can be formed from a disjunction of existing classes, e.g., $C \sqcup D$. This corresponds to a disjunction of unary predicates. When a disjunction occurs on the l.h.s. of a subclass axiom it simply becomes disjunction in the body of the corresponding rule:

$$C_1 \sqcup C_2 \sqsubseteq D \quad \equiv \quad D(x) \quad \leftarrow \quad C_1(x) \vee C_2(x)$$

This is easily transformed into a pair of def-Horn rules:

$$\begin{array}{lcl} D(x) & \leftarrow & C_1(x) \\ D(x) & \leftarrow & C_2(x) \end{array}$$

When a disjunction occurs on the r.h.s. of a subclass axiom it becomes a disjunction in the head of the corresponding rule, and this cannot be handled within the def-Horn framework.

Universal Restriction (DL $\forall$)

In a DL the universal quantifier can only be used in *restrictions*—expressions of the form $\forall P.C$ (see Section 4.2.1). This is equivalent to an FOL clause of the form $\forall y.P(x,y) \rightarrow C(y)$. P must be a single primitive property, but C may be a compound expression. Therefore, when a universal restriction occurs on the r.h.s. of a subclass axiom it becomes an implication in the head of the corresponding rule:

$$C \sqsubseteq \forall P.D \quad \equiv \quad (D(y) \quad \leftarrow \quad P(x,y)) \quad \leftarrow \quad C(x),$$

which is easily transformed into the standard def-Horn rule:

$$D(y) \quad \leftarrow \quad C(x) \wedge P(x,y).$$

When a universal restriction occurs on the l.h.s. of a subclass axiom it becomes an implication in the body of the corresponding rule. This cannot, in general, be mapped into def-Horn as it would require negation in a rule body.

Existential Restriction (DL $\exists$)

In a DL, the existential quantifier (like the universal quantifier) can only be used in restrictions of the form $\exists P.C$. This is equivalent to an FOL clause of the form $\exists y.P(x,y) \wedge C(y)$. P must be a single primitive property, but C may be a compound expression.

When an existential restriction occurs on the l.h.s. of a subclass axiom, it becomes a conjunction in the body of a standard def-Horn rule:

$$\exists P.C \sqsubseteq D \quad \equiv \quad D(x) \quad \leftarrow \quad P(x,y) \wedge C(y).$$

When an existential restriction occurs on the r.h.s. of a subclass axiom, it becomes a conjunction in the head of the corresponding rule, with a variable that is existentially quantified. This cannot be handled within the def-Horn framework.

Negation and Cardinality Restrictions (DL $\neg$, $\geqslant$ and $\leqslant$)

These constructors cannot, in general, be mapped into def-Horn. The case of negation is obvious as negation is not allowed in either the head or body of a def-Horn rule. Cardinality restrictions correspond to assertions of variable equality and inequality in FOL, and this is again outside of the def-Horn framework.

In some cases, however, it would be possible to simplify the DL expression using the usual rewriting tautologies of FOL in order to eliminate the offending operator(s). For example, negation can always be pushed inwards by using a combination of De Morgan's laws and equivalences such as $\neg\exists P.C \equiv \forall P.\neg C$ and $\neg\geqslant n\,P \equiv \leqslant (n-1)\,P$ [BCM$^+$03]. Further simplifications are also possible, e.g., using the equivalences $C \sqcup \neg C \equiv \top$, and $\forall P.\top \equiv \top$. For the sake of simplicity, however, we will assume that DL expressions are in a canonical form where all relevant simplifications have been carried out.

4.2.4 Defining DHL via a Recursive Mapping from DL to def-Horn

As we saw in Section 4.2.3, some DL constructors (conjunction and universal restriction) can be mapped to the heads of rules whenever they occur on the r.h.s. of an inclusion axiom, while some DL constructors (conjunction, disjunction and existential restriction) can be mapped to the bodies of rules whenever they occur on the l.h.s. of an inclusion axiom. This naturally leads to the definition of two DL languages, classes from which can be mapped into the head or body of LP rules; we will refer to these two languages as $\mathcal{L}_h$ and $\mathcal{L}_b$ respectively.

The syntax of the two languages is defined as follows. In both languages an atomic name A is a class, and if C and D are classes, then $C \sqcap D$ is also a class. In $\mathcal{L}_h$, if C is a class and R is a property, then $\forall R.C$ is also a class, while in $\mathcal{L}_b$, if D, C are classes and R is a property, then $C \sqcup D$ and $\exists R.C$ are also classes.

Using the mappings from Section 4.2.3, we can now define a recursive mapping function $\mathcal{T}$ which takes a DL axiom of the form $C \sqsubseteq D$, where C is an $\mathcal{L}_b$-class and D is an $\mathcal{L}_h$-class, and maps it into an LP rule of the form $A \leftarrow B$. The mapping is defined as follows:

$$
\begin{array}{lcl}
\mathcal{T}(C \sqsubseteq D) & \longrightarrow & Th(D,y) \leftarrow Tb(C,y) \\
Th(A,x) & \longrightarrow & A(x) \\
Th((C \sqcap D),x) & \longrightarrow & Th(C,x) \wedge Th(D,x) \\
Th((\forall R.C),x) & \longrightarrow & Th(C,y) \leftarrow R(x,y) \\
Tb(A,x) & \longrightarrow & A(x) \\
Tb((C \sqcap D),x) & \longrightarrow & Tb(C,x) \wedge Tb(D,x) \\
Tb((C \sqcup D),x) & \longrightarrow & Tb(C,x) \vee Tb(D,x) \\
Tb((\exists R.C),x) & \longrightarrow & R(x,y) \wedge Tb(C,y)
\end{array}
$$

where A is an atomic class name, C and D are classes, R is a property and x, y are variables, with y being a "fresh" variable, i.e., one that has not previously been used.

As we saw in Section 4.2.3, rules of the form $(H \wedge H') \leftarrow B$ are rewritten as two rules $H \leftarrow B$ and $H' \leftarrow B$; rules of the form $(H \leftarrow H') \leftarrow B$ are rewritten as $H \leftarrow (B \wedge H')$; and rules of the form $H \leftarrow (B \vee B')$ are rewritten as two rules $H \leftarrow B$ and $H \leftarrow B'$.

For example, $\mathcal{T}$ would map the DL axiom

$$A \sqcap \exists R.C \sqsubseteq B \sqcap \forall P.D$$

into the LP rule

$$B(x) \wedge (D(z) \leftarrow P(x,z)) \leftarrow A(x) \wedge R(x,y) \wedge C(x)$$

which is rewritten as the pair of rules

$$\begin{array}{lcl} B(x) & \leftarrow & A(x) \wedge R(x,y) \wedge C(x) \\ D(z) & \leftarrow & A(x) \wedge R(x,y) \wedge C(x) \wedge P(x,z). \end{array}$$

We call $\mathcal{L}$ the intersection of $\mathcal{L}_h$ and $\mathcal{L}_b$, i.e., the language where an atomic name A is a class, and if C and D are classes, then $C \sqcap D$ is also a class. We then extend $\mathcal{T}$ to deal with axioms of the form $C \equiv D$, where C and D are both $\mathcal{L}$-classes:

$$\mathcal{T}(C \equiv D) \quad \longrightarrow \quad \left\{ \begin{array}{l} \mathcal{T}(C \sqsubseteq D) \\ \mathcal{T}(D \sqsubseteq C) \end{array} \right.$$

As we saw in Section 4.2.2, range and domain axioms $\top \sqsubseteq \forall P.D$ and $\top \sqsubseteq \forall P^{-}.D$ are mapped into def-Horn rules of the form $D(y) \leftarrow P(x,y)$ and $D(x) \leftarrow P(x,y)$ respectively. Moreover, class-instance and instance-property-instance axioms $a : D$ and $\langle a,b \rangle : P$ are mapped into def-Horn facts (i.e., rules with empty bodies) of the form $D(a)$ and $P(a,b)$ respectively. We therefore extend $\mathcal{T}$ to deal with these axioms in the case that D is an $\mathcal{L}_h$-class:

$$\begin{array}{lcll} \mathcal{T}(\top \sqsubseteq \forall P.D) & \longrightarrow & Th(D,y) & \leftarrow \quad P(x,y) \\ \mathcal{T}(\top \sqsubseteq \forall P^{-}.D) & \longrightarrow & Th(D,x) & \leftarrow \quad P(x,y) \\ \mathcal{T}(a : D) & \longrightarrow & Th(D,a) & \\ \mathcal{T}(\langle a,b \rangle : P) & \longrightarrow & P(a,b) & \end{array}$$

where x, y are variables and a, b are constants.

Finally, we extend $\mathcal{T}$ to deal with the property axioms discussed in Section 4.2.2:

$$\begin{array}{lcl} \mathcal{T}(P \sqsubseteq Q) & \longrightarrow & Q(x,y) \leftarrow P(x,y) \\ \mathcal{T}(P \equiv Q) & \longrightarrow & \left\{ \begin{array}{l} Q(x,y) \leftarrow P(x,y) \\ P(x,y) \leftarrow Q(x,y) \end{array} \right. \\ \mathcal{T}(P \equiv Q^{-}) & \longrightarrow & \left\{ \begin{array}{l} Q(x,y) \leftarrow P(y,x) \\ P(y,x) \leftarrow Q(x,y) \end{array} \right. \\ \mathcal{T}(P^{+} \sqsubseteq P) & \longrightarrow & P(x,z) \leftarrow P(x,y) \wedge P(y,z) \end{array}$$

Definition 4.1 (Description Horn Logic) *A* Description Horn Logic (DHL) *ontology is a set of DHL axioms of the form $C \sqsubseteq D$, $A \equiv B$, $\top \sqsubseteq \forall P.D$, $\top \sqsubseteq \forall P^{-}.D$, $P \sqsubseteq Q$, $P \equiv Q$, $P \equiv Q^{-}$, $P^{+} \sqsubseteq P$, $a : D$ and $\langle a,b \rangle : P$, where C is an $\mathcal{L}_b$-class, D is an $\mathcal{L}_h$-class, A, B are $\mathcal{L}$-classes, P, Q are properties and a, b are individuals.*

Using the relationships of (full) DL to FOL [Bor96, BCM^{+}03], it is straightforward to show the following.

Theorem 4.1 (Translation Semantics) *The mapping $\mathcal{T}$ preserves semantic equivalence. Let $\mathcal{K}$ be a DHL ontology and $\mathcal{H}$ be the def-Horn ruleset that results from applying the mapping $\mathcal{T}$ to all the axioms in $\mathcal{K}$. Then $\mathcal{H}$ is logically equivalent to $\mathcal{K}$ w.r.t. the semantics of FOL — $\mathcal{H}$ has the same set of models and entailed conclusions as $\mathcal{K}$.*

DHL can, therefore, be viewed alternatively and precisely as an expressive fragment of def-Horn—i.e., as the *range* of $\mathcal{T}(\mathrm{DHL})$.

4.2.5 Expressive Power of DHL

Although the asymmetry of DHL (w.r.t. classes on different sides of axioms) makes it rather unusual by DL standards, it is easy to see that it includes (the OWL subset of) RDFS,[14] as well as that part of OWL which corresponds to a simple frame language.

As far as RDFS is concerned, we saw in Section 4.2.2 that RDFS statements are equivalent to DL axioms of the form $C \sqsubseteq D$, $\top \sqsubseteq \forall P.C$, $\top \sqsubseteq \forall P^{-}.C$, $P \sqsubseteq Q$, $a : D$ and $\langle a, b\rangle : P$, where C, D are classes, P, Q are properties and a, b are individuals. Given that all RDFS classes are $\mathcal{L}$-classes (they are atomic class names), a set of DL axioms corresponding to RDFS statements would clearly satisfy the above definition of a DHL ontology.

DHL also includes the subset of OWL corresponding to simple frame language axioms, i.e., axioms defining a primitive hierarchy of classes, where each class is defined by a frame. A frame specifies the set of subsuming classes and a set of slot constraints. This corresponds very neatly to a set of DL axioms of the form $A \sqsubseteq C$, where C is an $\mathcal{L}_h$-class.

Moreover, DHL supports the extension of this language to include equivalence of conjunctions of atomic classes, and axioms corresponding to OWL transitive property, and inverse property statements.

4.2.6 Defining DLP

Definition 4.2 (Description Logic Programs) *We say that a def-LP $\mathcal{RP}$ is a* Description Logic Program (DLP) *when it is the LP-correspondent of some DHL ruleset $\mathcal{RH}$.*

A DLP is directly defined as the LP-correspondent of a def-Horn ruleset that results from applying the mapping $\mathcal{T}$. Semantically, a DLP is thus the f-weakening of that DHL ruleset. The DLP expressive class is thus the expressive f-subset of DHL. By Theorem 4.1, DLP can, therefore, be viewed alternatively and precisely as an expressive subset of DL, not just of def-Horn.

In summary, expressively DLP is contained in DHL which in turn is contained in the expressive intersection of DL and Horn.

4.3 Inferencing

One of the motivations for this work is to enable some fragment of DL inferencing to be performed by LP engines. In this section we will discuss the kinds of inference typically of interest in DL and LP, and how they can be represented in each other, i.e., in LP and DL respectively. Although the emphasis is on performing DL inferencing, via our mapping translation, using an LP reasoning engine, the reverse mapping can be used in order to perform LP inferencing using a DL reasoning engine. In particular, we will show how inferencing in (the DHL fragment of) DL can be reduced, via our translation, to inferencing in LP; and how vice versa, inferencing in (the DLP fragment of) LP can be reduced to inferencing in DL.

In a DL reasoning system, several different kinds of query are typically supported w.r.t. a knowledge base $\mathcal{K}$. These include queries about classes:

1. class-instance membership queries: given a class C,
   (a) ground: determine whether a given individual a is an instance of C;
   (b) open: determine all the individuals in $\mathcal{K}$ that are instances of C;

[14] By the OWL subset of RDFS, we mean that part of RDFS which is included in OWL DL.

(c) "all-classes": given an individual a, determine all the (named) classes in $\mathcal{K}$ that a is an instance of;

2. class subsumption queries: i.e., given classes C and D, determine if C is a subclass of D w.r.t. $\mathcal{K}$;

3. class hierarchy queries: i.e., given a class C return all/most-specific (named) superclasses of C in $\mathcal{K}$ and/or all/most-general (named) subclasses of C in $\mathcal{K}$;

4. class satisfiability queries, i.e., given a class C, determine if C is satisfiable (consistent) w.r.t. $\mathcal{K}$.

In addition, there are similar queries about properties: property-instance membership, property subsumption, property hierarchy, and property satisfiability. We will call $\mathcal{QDL}$ the language defined by the above kinds of DL queries.

In LP reasoning engines, there is one basic kind of query supported w.r.t. a ruleset $\mathcal{R}$: atom queries. These include:

1. ground: determine whether a ground atom A is entailed;

2. open (ground is actually a special case of this): determine, given an atom A (in which variables may appear), all the tuples of variable bindings (substitutions) for which the atom is entailed.

We call $\mathcal{QLP}$ the language defined by the above kinds of LP queries.

Next, we discuss how to reduce $\mathcal{QDL}$ querying in (the DHL fragment of) DL to $\mathcal{QLP}$ querying in (the DLP fragment of) LP using the mapping $\mathcal{T}$. We will assume that $\mathcal{R}$ is a ruleset derived from a DL knowledge base $\mathcal{K}$ via $\mathcal{T}$, and that all $\mathcal{QDL}$ queries are w.r.t. $\mathcal{K}$.

$\mathcal{QLP}$ (ground or open) atom queries can be used to answer $\mathcal{QDL}$ (ground or open) class-instance membership queries when the class is an $\mathcal{L}_h$-class, i.e., a is an instance of C iff $\mathcal{R}$ entails $\mathcal{T}(a : C)$. When C is an atomic class name, the mapping leads directly to a $\mathcal{QLP}$ atom query. When C is a conjunction, the result is a conjunction of $\mathcal{QLP}$ atom queries, i.e., a is an instance of $C \sqcap D$ iff $\mathcal{R}$ entails $\mathcal{T}(a : C)$ *and* $\mathcal{R}$ entails $\mathcal{T}(a : D)$. When C is a universal restriction, the mapping $\mathcal{T}(a : \forall P.C)$ gives $\mathcal{T}(C, y) \leftarrow P(a, y)$. This can be transformed into a $\mathcal{QLP}$ atom query using a simple kind of Skolemisation, i.e., y is replaced with a constant b, where b is new in $\mathcal{R}$, and we have a is an instance of $\forall P.C$ iff $\mathcal{R} \cup \{P(a, b)\}$ entails $\mathcal{T}(b : C)$.

The case of property-instance membership queries is trivial as all properties are atomic: $\langle a, b\rangle$ is an instance of P iff $\mathcal{R}$ entails $P(a, b)$.

Complete information about class-instance relationships, to answer open or "all-classes" class-instance queries, can then be obtained via class-instance queries about all possible combinations of individuals and classes in $\mathcal{K}$.[15] (Note that the set of named individuals and classes is known, and its size is worst-case linear in the size of the knowledge/rule base.)

For $\mathcal{L}_h$-classes, $\mathcal{QDL}$ class subsumption queries can be reduced to $\mathcal{QLP}$ using a similar technique to class-instance membership queries, i.e., C is a subclass of D iff $\mathcal{R} \cup \{\mathcal{T}(a : C)\}$ entails $\mathcal{T}(a : D)$, for a new in $\mathcal{R}$. For $\mathcal{QDL}$ property subsumption queries, P is a subproperty of Q iff $\mathcal{R} \cup P(a, b)$ entails $Q(a, b)$, for a, b new in $\mathcal{R}$.

Complete information about the class hierarchy can be obtained by computing the partial ordering of classes in $\mathcal{K}$ based on the subsumption relationship.

In the DHL (and DLP) fragment, determining class/property satisfiability is a non-issue as, with the expressive power at our disposal in def-Horn, it is impossible to make a class or a property unsatisfiable.

[15] More efficient algorithms would no doubt be used in practice.

Now let us consider the reverse direction from $\mathcal{QLP}$ to $\mathcal{QDL}$. In the DLP fragment of LP, every predicate is either unary or binary. Every atom query can thus be viewed as about either a named class or a property. Also, generally in LP, any open atom query is formally reducible to a set of ground atom queries — one for each of its instantiations. Thus $\mathcal{QLP}$ is reducible to class-instance and property-instance membership queries in DL.

To recap, we have shown the following.

Theorem 4.2 (Inferencing Inter-operability) *For $\mathcal{L}_h$-classes, $\mathcal{QDL}$ querying in (the DHL fragment of) DL is reducible to $\mathcal{QLP}$ querying in (the DLP fragment of) LP, and vice versa.*

All of these queries can be reduced to one basic inference task: that of determining knowledge base satisfiability. An individual i is an instance of a class C iff adding $i : \neg C$ to $\mathcal{K}$ makes $\mathcal{K}$ unsatisfiable, and complete information about class-instance relationships could be obtained by applying this test to all possible combinations of individuals and classes in $\mathcal{K}$.[16] A class C is unsatisfiable iff adding $i : \neg C$ to $\mathcal{K}$ (for some individual i not already occurring in $\mathcal{K}$) makes $\mathcal{K}$ unsatisfiable, and C is a subclass of D iff $D \sqcup \neg C$ is unsatisfiable. Finally, complete information about the class hierarchy can be obtained by computing the partial ordering of classes in $\mathcal{K}$ based on the subsumption relationship.

The above queries can also be applied to properties, but are trickier to answer because DLs do not typically support property negation. Therefore, it is not possible to determine if a tuple $\langle i_1, i_2 \rangle$ is an instance of a property P by adding $\langle i_1, i_2 \rangle : \neg P$ to $\mathcal{K}$. The same effect can be achieved, however, by checking if adding $i : \exists P.\{j\}$ to $\mathcal{K}$ makes it unsatisfiable. Similarly, P is a subproperty of Q iff $\exists P.\{k\} \sqcap \neg \exists Q.\{k\}$ is unsatisfiable w.r.t. $\mathcal{K}$.

4.4 Discussion

We have shown how to interoperate, semantically and inferentially, between Rules (in particular Logic Programs) and DLs, two Knowledge Representation (KR) paradigms widely used in ontology engineering. We have begun by studying two new KRs, Description Logic Programs (DLP), which is defined by the expressive intersection of the two approaches, and the closely related Description Horn Logic (DHL).

We have shown that DLP (or DHL) can capture a significant fragment of OWL, including the whole of the OWL subset of RDFS, simple frame axioms and more expressive property axioms. Many of the ontologies in the OWL ontology library are inside this fragment of OWL. An immediate result of this work is that LP engines could be used for reasoning with these ontologies and for reasoning with (possibly very large numbers of) facts, such as web page annotations, that use vocabulary from these ontologies.

This work represents only a first step in realising a more complete interoperability between rules and ontologies, and the layering of rules on top of ontology languages in the Semantic Web "stack". We believe, however, that our study of the expressive intersection will provide a firm foundation for future investigations of more expressive languages up to and including the expressive union of rules and ontologies.

[16] More efficient algorithms would no doubt be used in practice.

5 SWRL: extending OWL with Rules

Ian Horrocks and Peter F. Patel-Schneider

Much of the material in this section first appeared in [HPS04]: Ian Horrocks and Peter F. Patel-Schneider: A proposal for an OWL Rules Language. *Proc. of WWW 2004, pages 723-731. ACM, 2004.*

The OWL Web Ontology Language [SWM03] adds considerable expressive power to the Semantic Web. However, for a variety of reasons (see `http://lists.w3.org/Archives/Public/www-webont-wg/`), including retaining the decidability of key inference problems in OWL DL and OWL Lite, OWL has expressive limitations. These restrictions can be onerous in some application domains, for example in describing web services, where it may be necessary to relate inputs and outputs of composite processes to the inputs and outputs of their component processes [The03].

Many of the limitations of OWL stem from the fact that, while the language includes a relatively rich set of class constructors, the language provided for talking about properties is much weaker. In particular, there is no composition constructor, so it is impossible to capture relationships between a composite property and another (possibly composite) property. The standard example here is the obvious relationship between the composition of the "parent" and "brother" properties and the "uncle" property.

One way to address this problem would be to extend OWL with a more powerful language for describing properties. For example, a decidable extension of the description logics underlying OWL DL to include the use of composition in subproperty axioms has already been investigated [HS03]. In order to maintain decidability, however, the usage of the constructor is limited to axioms of the form $P \circ Q \sqsubseteq P$, i.e., axioms asserting that the composition of two properties is a subproperty of one of the composed properties. This means that complex relationships between composed properties cannot be captured—in fact even the relatively simple "uncle" example cannot not be captured (because "uncle" is not one of "parent" or "brother").

An alternative way to overcome some of the expressive restrictions of OWL is to extend it with some form of "rules language". In fact adding rules to description logic based knowledge representation languages is far from being a new idea. Several early description logic systems, e.g., Classic [PSMB$^+$91, BPS94], included a rule language component. In these systems, however, rules were given a weaker semantic treatment than axioms asserting sub- and super-class relationships; they were only applied to individuals, and did not affect class based inferences such as the computation of the class hierarchy. More recently, the CARIN system integrated rules with a description logic in such a way that sound and complete reasoning was still possible [LR98]. This could only be achieved, however, by using a rather weak description logic (*much* weaker than OWL), and by placing severe syntactic restrictions on the occurrence of description logic terms in the (heads of) rules. Similarly, the DLP language proposed in [GHVD03b] is based on the intersection of a description logic with horn clause rules; the result is obviously a decidable language, but one that is necessarily less expressive than either the description logic or rules language from which it is formed.

Here we show how a simple form of Horn-style rules can be added to the OWL language in a syntactically and semantically coherent manner, the basic idea being to add such rules as a new kind of axiom in OWL DL. We show (in Section 5.2) how the OWL abstract syntax in the OWL Semantics and Abstract Syntax document [PSHH03] can be extended to provide a formal syntax for these rules, and (in Section 5.3) how the direct OWL model-theoretic semantics for OWL DL can be extended to provide a formal meaning for OWL ontologies including rules written in this abstract syntax. We will also show (in Section 5.4) how OWL's XML and RDF/XML presentation syntaxes can be modified to deal with

the proposed rules.

The extended language, which has been called the Semantic Web Rules Language (SWRL) [HPSB+04], is considerably more powerful than either OWL DL or Horn rules alone, and in Section 5.5 we will show that the key inference problems (e.g., ontology consistency) for SWRL are undecidable.

5.1 Overview

The basic idea of SWRL is to extend OWL DL with a form of rules while maintaining maximum backwards compatibility with OWL's existing syntax and semantics. To this end, SWRL adds a new kind of axiom to OWL DL, namely Horn clause rules, extending the OWL abstract syntax and the direct model-theoretic semantics for OWL DL [PSHH03] to provide a formal semantics and syntax for OWL ontologies including such rules.

SWRL rules are of the form of an implication between an antecedent (body) and consequent (head). The informal meaning of a rule can be read as: whenever (and however) the conditions specified in the antecedent hold, then the conditions specified in the consequent must also hold.

Multiple atoms in an antecedent are treated as a conjunction. An empty antecedent is thus treated as trivially true (i.e. satisfied by every interpretation), so the consequent must also be satisfied by every interpretation.

Multiple atoms in a consequent are treated as separate consequences, i.e., they must all be satisfied. In keeping with the usual treatment in rules, an empty consequent is treated as trivially false (i.e., not satisfied by any extended interpretation). Such rules are satisfied if and only if the antecedent is not satisfied by any extended interpretation. Note that rules with multiple atoms in the consequent could easily be transformed into multiple rules each with an atomic consequent.

It is easy to see that OWL DL becomes undecidable when extended in this way as rules can be used to simulate role value maps [Sch89] and make it easy to encode known undecidable problems as an SWRL ontology consistency problem (see Section 5.5).

5.2 Abstract Syntax

The syntax for SWRL abstracts from any exchange syntax for OWL and thus facilitates access to and evaluation of the language; it extends the abstract syntax of OWL described in the OWL Semantics and Abstract Syntax document [PSHH03].

Like the OWL abstract syntax, the abstract syntax for rules is specified by means of a version of Extended BNF, very similar to the Extended BNF notation used for XML [Tim00]. In this notation, terminals are quoted; non-terminals are bold and not quoted. Alternatives are either separated by vertical bars (|) or are given in different productions. Components that can occur at most once are enclosed in square brackets ([...]); components that can occur any number of times (including zero) are enclosed in braces ({...}). Whitespace is ignored in the productions given here.

Names in the abstract syntax are RDF URI references [KC03]. These names may be abbreviated into qualified names, using one of the following namespace names:

rdf `http://www.w3.org/1999/02/22-rdf-syntax-ns#`
rdfs `http://www.w3.org/2000/01/rdf-schema#`
xsd `http://www.w3.org/2001/XMLSchema#`
owl `http://www.w3.org/2002/07/owl#`

The meaning of each construct in the abstract syntax for rules is informally described when it is introduced. The formal meaning of these constructs is given in Section 5.3 via an extension of the OWL DL model-theoretic semantics [PSHH03].

5.2.1 Rules

From the OWL Semantics and Abstract Syntax document [PSHH03], an OWL ontology in the abstract syntax contains a sequence of annotations, axioms, and facts. Axioms may be of various kinds, for example, subClass axioms and equivalentClass axioms. SWRL extends axioms to also allow rule axioms, by adding the production:

axiom ::= rule

Thus a SWRL ontology could contain a mixture of rules and other OWL DL constructs, including ontology annotations, axioms about classes and properties, and facts about OWL individuals, as well as the rules themselves.

A rule axiom consists of an antecedent (body) and a consequent (head), each of which consists of a (possibly empty) set of atoms. Just as for class and property axioms, rule axioms can also have annotations; a rule axiom may also be assigned a URIreference which could, e.g., be used to identify it.

rule ::= 'Implies(' [URIreference] { annotation } antecedent consequent ')'
antecedent ::= 'Antecedent(' { atom } ')'
consequent ::= 'Consequent(' { atom } ')'

Informally, a rule may be read as meaning that if the antecedent holds (is "true"), then the consequent must also hold. An empty antecedent is treated as trivially holding (true), and an empty consequent is treated as trivially not holding (false). Non-empty antecedents and consequents hold iff all of their constituent atoms hold. As mentioned above, rules with multiple consequents could easily transformed into multiple rules each with a single atomic consequent.

Atoms in rules can be of the form C(x), D(z), P(x,y), Q(x,z), sameAs(x,y), differentFrom(x,y), or builtIn(b,z_1,...,z_n) where C is an OWL DL description, D is and OWL DL data range, P is an OWL DL *individual-valued* Property, Q is an OWL DL *data-valued* Property x,y are either variables or OWL individuals, and z,z_1,...,z_n are either variables or OWL data literals. In the context of OWL Lite, descriptions in atoms of the form C(x) may be restricted to class names.

atom ::= description '(' i-object ')'
| dataRange '(' d-object ')'
| individualvaluedPropertyID '(' i-object i-object ')'
| datavaluedPropertyID '(' i-object d-object ')'
| sameAs '(' i-object i-object ')'
| differentFrom '(' i-object i-object ')'
| builtIn '(' builtinID d-object ')'

Informally, an atom C(x) holds if x is an instance of the class description C, an atom D(z) holds if z is a value in the dataRange D, an atom P(x,y) (resp. Q(x,z)) holds if x is related to y (z) by property P (Q), an atom sameAs(x,y) holds if x is interpreted as the same object as y, an atom differentFrom(x,y) holds if x and y are interpreted as different objects, and an atom builtIn(b,z_1,...,z_n) holds if (z_1,...,z_n) is in the extension of the built-in predicate b. A wide range of built-in predicates are described in the SWRL specification [HPSB$^+$04].

Atoms may refer to individuals, data literals, individual variables or data variables. Variables are treated as universally quantified, with their scope limited to a given rule. As usual, only variables that occur in the antecedent of a rule may occur in the consequent (a condition usually referred to as "safety").

i-object ::= i-variable | individualID
d-object ::= d-variable | dataLiteral

i-variable ::= 'I-variable(' URIreference ')'
d-variable ::= 'D-variable(' URIreference ')'

5.2.2 Human Readable Syntax

While the abstract Extended BNF syntax is consistent with the OWL specification, and is useful for defining XML and RDF serialisations, it is rather verbose and not particularly easy to read. In the following we will, therefore, often use a relatively informal "human readable" form similar to that used in many published works on rules.

In this syntax, a rule has the form:

$$\mathsf{antecedent} \rightarrow \mathsf{consequent},$$

where both antecedent and consequent are conjunctions of atoms written $a_1 \wedge \ldots \wedge a_n$. Variables are indicated using the standard convention of prefixing them with a question mark (e.g., $?x$). Using this syntax, a rule asserting that the composition of parent and brother properties implies the uncle property would be written:

$$\mathsf{parent}(?a, ?b) \wedge \mathsf{brother}(?b, ?c) \rightarrow \mathsf{uncle}(?a, ?c). \quad (1)$$

If John has Mary as a parent and Mary has Bill has a brother, then this rule requires that John has Bill as an uncle.

5.3 Direct Model-Theoretic Semantics

The model-theoretic semantics for SWRL is a straightforward extension of the semantics for OWL DL given in [PSHH03]. The basic idea is that we define *bindings*—extensions of OWL interpretations that also map variables to elements of the domain in the usual manner. A rule is satisfied by an interpretation iff every binding that satisfies the antecedent also satisfies the consequent. The semantic conditions relating to axioms and ontologies are unchanged, so an interpretation satisfies an ontology iff it satisfies every axiom (including rules) and fact in the ontology.

5.3.1 Interpreting Rules

From the OWL Semantics and Abstract Syntax document [PSHH03] we recall that an abstract OWL interpretation is a tuple of the form

$$\mathcal{I} = \langle R, EC, ER, L, S, LV \rangle,$$

where R is a set of resources, $LV \subseteq R$ is a set of literal values, EC is a mapping from classes and datatypes to subsets of R and LV respectively, ER is a mapping from properties to binary relations on R, L is a mapping from typed literals to elements of LV, and S is a mapping from individual names to elements of $EC(\mathsf{owl:Thing})$.

Given an abstract OWL interpretation $\mathcal{I}$, a binding $B(\mathcal{I})$ is an abstract OWL interpretation that extends $\mathcal{I}$ such that S maps i-variables to elements of $EC(\mathsf{owl:Thing})$ and L maps d-variables to elements of LV respectively. An atom is satisfied by a binding $B(\mathcal{I})$ under the conditions given in Table 6, where C is an OWL DL description, D is an OWL DL data range, P is an OWL DL *individual-valued* Property, Q is an OWL DL *data-valued* Property, B is a built-in predicate, x, y are variables or OWL individuals, $z, z_1, \ldots, z_n$ are variables or OWL data values, and Ext is a mapping from built-in predicates to a subset of $\bigcup_{1\ldots n} LV^n$. Note that this interpretation of the built-in predicates is very flexible

Atom	Condition on Interpretation
$C(x)$	$S(x) \in EC(C)$
$D(z)$	$S(z) \in EC(D)$
$P(x, y)$	$\langle S(x), S(y) \rangle \in ER(P)$
$Q(x, z)$	$\langle S(x), L(z) \rangle \in ER(Q)$
sameAs(x, y)	$S(x) = S(y)$
differentFrom(x, y)	$S(x) \neq S(y)$
builtIn$(B, z_1, \ldots, z_n)$	$\langle S(z_1), \ldots, S(z_n) \rangle \in Ext(B)$

Table 6: Interpretation Conditions

and allows, e.g., variable arity predicates and using predicates with the wrong number of arguments: if $\langle S(z_1), \ldots, S(z_n) \rangle$ is not in the extension of the built-in predicate, then the atom is simply unsatisfiable.

A binding $B(\mathcal{I})$ satisfies an antecedent A iff A is empty or $B(\mathcal{I})$ satisfies every atom in A. A binding $B(\mathcal{I})$ satisfies a consequent C iff C is not empty and $B(\mathcal{I})$ satisfies every atom in C. A rule is satisfied by an interpretation $\mathcal{I}$ iff for every binding B such that $B(\mathcal{I})$ satisfies the antecedent, $B(\mathcal{I})$ also satisfies the consequent.

The semantic conditions relating to axioms and ontologies are unchanged. In particular, an interpretation satisfies an ontology iff it satisfies every axiom (including rules) and fact in the ontology; an ontology is consistent iff it is satisfied by at least one interpretation; an ontology O_2 is entailed by an ontology O_1 iff every interpretation that satisfies O_1 also satisfies O_2.

5.3.2 Example

Consider, for example, the "uncle" rule (1) from Section 5.2.2. Assuming that parent, brother and uncle are *individualvaluedPropertyID*s, then given an interpretation $\mathcal{I} = \langle R, EC, ER, L, S, LV \rangle$, a binding $B(\mathcal{I})$ extends S to map the variables $?a$, $?b$, and $?c$ to elements of $EC(\mathsf{owl:Thing})$; we will use a, b, and c respectively to denote these elements. The antecedent of the rule is satisfied by $B(\mathcal{I})$ iff $(a, b) \in ER(\mathsf{parent})$ and $(b, c) \in ER(\mathsf{brother})$. The consequent of the rule is satisfied by $B(\mathcal{I})$ iff $(a, c) \in ER(\mathsf{uncle})$. Thus the rule is satisfied by $\mathcal{I}$ iff for every binding $B(\mathcal{I})$ such that $(a, b) \in ER(\mathsf{parent})$ and $(b, c) \in ER(\mathsf{brother})$, then it is also the case that $(a, c) \in ER(\mathsf{uncle})$, i.e.:

$$\forall a, b, c \in EC(\mathsf{owl:Thing}).$$
$$((a, b) \in ER(\mathsf{parent}) \wedge (b, c) \in ER(\mathsf{brother})) \rightarrow (a, c) \in ER(\mathsf{uncle})$$

5.4 SWRL Concrete Syntax

SWRL has been provided with both an XML and an RDF Concrete Syntax. The XML Concrete Syntax is a combination of the OWL Web Ontology Language XML Presentation Syntax [HEPS03] with the RuleML XML syntax.[17] This has several advantages:

- arbitrary OWL classes (e.g., descriptions) can be used as predicates in rules;
- rules and ontology axioms can be freely mixed;

[17] http://www.ruleml.org/

- interoperability between OWL and RuleML is simplified, facilitating tool development/adaption and the extension of SWRL with additional features from RuleML.

A further advantage of extending OWL's presentation syntax is that the existing XSLT stylesheet[18] can be extended to provide a mapping to RDF graphs that extends the OWL RDF/XML exchange syntax. Full details of both the XML and the RDF concrete syntax can be found in the SWRL member submission [HPSB+04].

5.5 The Power of Rules

In OWL, the only relationship that can be asserted between properties is subsumption between atomic property names, e.g., asserting that hasFather is a subPropertyOf hasParent. In Section 5.2.2 we have already seen how a rule can be used to assert more complex relationships between properties. While this increased expressive power is clearly very useful, it is easy to show that it leads to the undecidability of key inference problems, in particular ontology consistency.

For extensions of languages such as OWL DL, the undecidability of the consistency problem is often proved by showing that the extension makes it possible to encode a known undecidable domino problem [Ber66] as an ontology consistency problem. In particular, it is well known that such languages only need the ability to represent an infinite 2-dimensional grid in order for consistency to become undecidable [BS96, HST99]. With the addition of rules, such an encoding is trivial. For example, given two properties x-succ and y-succ, the rule:

$$\text{x-succ}(?a, ?b) \ \wedge \ \text{y-succ}(?b, ?c) \ \wedge \ \text{y-succ}(?a, ?d) \ \wedge \ \text{x-succ}(?d, ?e) \quad \rightarrow \quad \text{sameAs}(?c, ?e),$$

along with the assertion that every grid node is related to exactly one other node by each of x-succ and y-succ, allows such a grid to be represented. This would be possible even without the use of the sameAs atom in the consequent—it would only be necessary to establish appropriate relationships with a "diagonal" property:

$$\text{x-succ}(?a, ?b) \wedge \text{y-succ}(?b, ?c) \rightarrow \text{diagonal}(?a, ?c)$$
$$\text{y-succ}(?a, ?d) \wedge \text{x-succ}(?d, ?e) \rightarrow \text{diagonal}(?a, ?e),$$

and additionally assert that every grid node is related to exactly one other node by diagonal.

SWRL rules seem to go beyond basic Horn clauses in allowing:

- conjunctive consequents;
- class descriptions as well as class names as predicates in class atoms; and
- equalities and inequalities.

On closer examination, however, it becomes clear that most of this is simply "syntactic sugar", and does not add to the power of the language.

In the case of conjunctive consequents, it is easy to see that these could be eliminated using the standard Lloyd-Topor transformation [Llo87b]. For example, a rule of the form

$$A \rightarrow C_1 \wedge C_2$$

[18] http://www.w3.org/TR/owl-xmlsyntax/owlxml2rdf.xsl

can be transformed into a semantically equivalent pair of rules

$$A \rightarrow C_1$$
$$A \rightarrow C_2.$$

In the case of class descriptions, it is easy to see that a description d can be eliminated from a rule simply by adding an OWL axiom that introduces a new class name and asserts that it is equivalent to d, e.g.,

$$\mathsf{EquivalentClasses}(D\ d).$$

The description can then be replaced with the name, here replacing the description d with class name D.

In the case of equality atoms, the sameAs property could easily be substituted with a "user defined" owl property called, for example, Eq. Such a property can be given the appropriate meaning using a rule of the form

$$\mathsf{Thing}(?x) \rightarrow \mathsf{Eq}(?x, ?x) \tag{2}$$

and by asserting that it is functional. It is easy to see that the interpretation of Eq corresponds to equality of elements in $EC(\mathsf{owl:Thing})$, i.e.,

$$\forall x, y \in EC(\mathsf{owl:Thing}).\langle x, y\rangle \in ER(\mathsf{Eq}) \iff x = y,$$

and that Eq could therefore be used instead of sameAs without changing the meaning of the ontology.

The case of inequalities is slightly more complex. An owl property called, for example, Neq, can be introduced and used to capture some of the meaning of the differentFrom property by adding a rule of the form

$$\mathsf{Eq}(?x, ?y) \wedge \mathsf{Neq}(?x, ?y) \rightarrow \mathsf{Nothing}(?x). \tag{3}$$

It is easy to see that the interpretation of Neq is disjoint from the interpretation of Eq, i.e.,

$$\forall x, y \in EC(\mathsf{owl:Thing}).\langle x, y\rangle \in ER(\mathsf{Neq}) \implies x \neq y,$$

and that this leads to the implicit rule

$$\mathsf{Neq}(?x, ?y) \rightarrow \mathsf{differentFrom}(?x, ?y).$$

Rule 3 shows that we could eliminate differentFrom when it occurs in the consequent of a rule simply by substituting Neq. Neq does not, however, fully capture the meaning of inequality, because there could be pairs of elements in $EC(\mathsf{owl:Thing})$ that are in the extension of neither Eq nor Neq, i.e., differentFrom does *not* imply Neq. As a result, we cannot use Neq to eliminate occurrences of differentFrom in the antecedent of a rule: in order to do so it would require Neq to be equivalent to the negation of Eq.

5.6 Discussion

SWRL is an extension to OWL to include a simple form of Horn-style rules. The main strengths of SWRL are its simplicity and its tight integration with the existing OWL language. As we have seen, SWRL extends OWL DL with the most basic kind of Horn rule (sweetened with a little "syntactic sugar"), plus built-in predicates for data values: ordinary predicates are limited to being OWL classes and properties (and so have a maximum arity of 2), there are no disjunctions or negations (of atoms), and no nonmonotonic features such as negation as failure or defaults. Moreover, rules are given a standard first order semantics. This facilitates the tight integration with OWL, with SWRL being defined as a syntactic and semantic extension of OWL DL.

6 Hybrid Integration of rules and DL-based ontologies

Jan Maluszynski

6.1 Motivation and Overview

In the SWRL approach, discussed above, integration of rules and ontologies is achieved by defining a new language, where rules may be used for defining new classes and new properties of the ontology. The new language obtained in that way is yet another ontology language, more expressive then OWL DL. It requires development of new reasoning techniques and new reasoners.

While ontologies are assumed to provide commonly shared conceptualization of a domain, there may be different application-specific rule programs for different applications in the domain. In that case rules would not define new classes or properties of the ontology, but rather some application-specific relations, while referring in the bodies to classes and properties defined by a given ontology. This is similar to defining views in a given relational database.

Such an approach is called *hybrid*. Thus in the hybrid approach the ontology remains unchanged and rules are built on top of ontologies. This makes possible integration of existing rule reasoner with existing ontology reasoner for reasoning in the hybrid language, rather than developing a new reasoner from scratch.

In Section 3 we mentioned several examples of homogeneous systems integrating LP rules with RDF and RDFS. These approaches focused on implementation issues, and were restricted to RDFS. This section surveys proposals for hybrid integration of different kinds of rules with different kinds of description logics.

More precisely the idea of the hybrid approach can be stated as follows. We consider an LP rule language $\mathcal{R}$ and an ontology language $\mathcal{S}$ based on a description logic. We assume that they share the alphabet of variables and the alphabet of individual constants, but their alphabets of predicate symbols are disjoint. We assume that both languages are supported by reasoners answering queries in the respective query languages, $\mathcal{Q}_\mathcal{R}$ and $\mathcal{Q}_\mathcal{S}$. Examples of such rule languages are Datalog with the least Herbrand model semantics, and function-free normal LP with answer set semantics, discussed in Section 2.3. An example of such an ontology language is OWL DL ($\mathcal{SHOIN}$).

A hybrid rule over $\mathcal{R}$ and $\mathcal{S}$ is a rule of the form

$$H \leftarrow B_1 \wedge \ldots \wedge B_m \wedge Q_1 \wedge \ldots \wedge Q_n$$

where, $m, n \geq 0$, H, B_i are literals, and Q_j are queries in the query language $\mathcal{Q}_\mathcal{S}$. A hybrid rule, where $m = n = 0$, is, as usual, called *fact*.

A *hybrid knowledge base* $K = (S, R)$ is a finite set S of DL axioms in the ontology language $\mathcal{S}$ and a finite set of hybrid rules R over $\mathcal{R}$ and $\mathcal{S}$, including non-DL atoms. By a DL component of a hybrid knowledge base we mean the set of all DL axioms, including the terminological axioms and assertions.

The semantics of a hybrid language is derived from the semantics of its components. A hybrid rule r is obtained from an LP rule r' by adding DL queries in the body. Intuitively, the latter are additional constraints on the use of the rule r' which have to be satisfied to draw the conclusions by r'. They may share variables and constants with the literals of r'. As the semantics of the DL component is not changed the semantics of the hybrid language is thus obtained by adaptation of the semantics of the rule component. For $\mathcal{R}$ being the language of definite logic programs the hybrid rules are formulae of FOL and their semantics can be formalized in terms of logical interpretations and models. For hybrid extensions of normal logic programs one has to refine the semantics of normal rules, e.g. the answer set

semantics. Reasoners for hybrid languages constructed by integration of reasoners of the components should be sound and complete with respect of this semantics.

Typically the reasoners are used for query answering. The query language of the hybrid rule language coincides syntactically with the query language $\mathcal{Q}_{\mathcal{R}}$ of the rule component. An answer to a query is an instance of the query entailed by the hybrid knowledge base. However, the predicates of a hybrid knowledge base are defined by hybrid rules with body constraints which must be satisfied when deriving answers.

For the hybrid integration of positive logic programs with a DL the following approach to query answering is possible [19]. (See the next section for an example). For a ground atomic query q use first a backward chaining reasoner of $\mathcal{R}$ for construction of a derivation of maximal length. The last element of such a derivation includes conjunction of the DL queries of all the rules applied in the derivation (since DL-queries cannot be resolved with the rules of the knowledge base). If this last element does not include non-DL queries we will call it a DL-constraint of q. In general rules are nondeterministic and q may have several derivations, generating several DL-constraints. q is entailed by the hybrid knowledge base (S, R) if the disjunction of all its DL-constraints is entailed by S. For checking this the DL reasoner of the DL $\mathcal{S}$ is to be used. This idea can be extended to answering non-ground hybrid queries.

Another approach to reasoning may be based on forward chaining, as proposed in [LR98].

6.2 Hybrid systems integrating rules and DL

6.2.1 $\mathcal{AL}$-log

The $\mathcal{AL}$-log language described in [DLNS98] is a hybrid integration of Datalog and the Description Logic $\mathcal{ALC}$. The $\mathcal{ALC}$ DL is a simple Description Logic admitting only the following class constructors (according to the terminology of Figure 4): `intersectionOf, unionOf, complementOf, someValuesFrom` and `allValuesFrom`. The DL queries in the bodies of the $\mathcal{AL}$-log rules are restricted to (ground or open) class-instance membership queries; property instance membership queries are not allowed. Moreover, the variables of the DL queries in the body of an $\mathcal{AL}$-log rule must also appear in the non-DL atoms of the body or in the head. Thus the DL queries are typing constraints on the variables.

$\mathcal{AL}$-queries are conjunctions of atomic formulae built with non-DL predicates. Query answering in $\mathcal{AL}$-log is decidable. The query answering algorithm described in [DLNS98] constructs first DL constraints for a given query using backward chaining (based on SLD resolution), and uses then an $\mathcal{ALC}$ tableau reasoner to check that the disjunction of the obtained DL constraints is entailed by the DL component of the $\mathcal{AL}$-log knowledge base.

We illustrate $\mathcal{AL}$-log by an example from [DLNS98], describing some university regulations.
The DL component of the example hybrid knowledge base includes

- the class names: `FP` Full Professor, `NFP` Non-Teaching Full Professor, `FM` Faculty Member, `St` Student, `Tp` Topic, `Co` Course, `AC` Advanced Course, and `BC` Basic Course.
- the property name `TC` Teaching
- the axioms
  $\texttt{FP} \sqsubseteq \texttt{FM}$ *Any full professor is a faculty member*
  $\texttt{NFP} \equiv \texttt{FP} \sqcap \neg\exists\, \texttt{TC.Co}$ *NFP is defined as a full professor that does not teach any course*

[19] This idea was suggested and proved correct and complete in [DLNS98] for hybrid integration of Datalog and the ALC Description Logic, discussed in the next section.

$\texttt{AC} \sqcup \texttt{BC} \equiv \texttt{Co}$ *The class of courses coincides with the union of advanced and basic courses*
$\texttt{AC} \sqcap \texttt{BC} \equiv \bot$ *Advanced Courses and Basic Courses are disjoint*
`john:FP` *John is a Full Professor*
mary :FP $(\sqcap \forall \texttt{TC.AC})$ *Mary is a Full Professor and all courses she teaches are advanced*
`paul:St` *Paul is a student*
`ai:AC` *Artificial Intelligence is an advanced course*
`kr:Tp` *Knowledge Representation is a topic*
`lp:Tp` *Logic Programming is a topic*
$\langle$`john,ai`$\rangle$`:TC` *John teaches Artificial Intelligence*

The rule component of the example hybrid knowledge base includes

- the predicates:
  `mayDoThesis(?x,?y)` *student* `?x` *may do the thesis with professor* `?y`,
  `curr(?x,?y)` *student* `?x` *had topic* `?y` *in his/her curriculum*
  `expert(?x,?y)` *professor* `?x` *is an expert on topic* `?y`
  `exam(?x,?y)` *student* `?x` *passed the exam on topic* `?y`
  `subject(?x,?y)` *course* `?x` *covers topic* `?y`

- the rules (the DL queries placed in separate lines):
  `curr(?x,?z)` $\leftarrow$ `exam(?x,?y)` $\wedge$ `subject(?y,?z)`
  $\wedge$ `?x:St` $\wedge$ `?y:Co` $\wedge$ `?z:Tp`
  student `?x` *had topic* `?z` *in his/her curriculum if he/she passed the exam in course* `?y` *covering topic* `?z`

  `mayDoThesis(?x,?y)` $\leftarrow$ `curr(?x,?z)` $\wedge$ `expert(?y,?z)`
  $\wedge$ `?x:St` $\wedge$ `?z:Tp` $\wedge$ `?y:(FM`$\sqcap\exists$`TC.AC)`
  student `?x` *may do thesis with professor* `?y` *if* `?x` *had topic* `?z` *in his/her curriculum,* `?y` *is an expert* `z`*, is a faculty member and teaches some advanced courses.*

  `mayDoThesis(?x,?y)` $\leftarrow$
  `?x:St` $\wedge$ `?y:NFP`
  student `?x` *may do thesis with professor* `?y` *if* `?y` *is non-teaching full professor.*

- The facts:
  `exam(paul,ai)` *Paul passed the exam on AI,*
  `subject(ai,kr)` *AI course covers Knowledge Representation,*
  `subject(ai,lp)` *AI course covers Logic Programming,*
  `expert(john,kr)` *John is an expert on Knowledge Representation,*
  `expert(mary,lp)` *Mary is an expert on Logic Programming*

We now discuss how the query `mayDoThesis(paul,mary)` would be answered by combination of rule reasoning and DL reasoning discussed in the previous section.

The following derivations can be constructed from the query by backward chaining

1. `mayDoThesis(paul,mary)`

   `curr(paul,?z)` $\wedge$
   `expert(mary,?z)` $\wedge$ `paul:St` $\wedge$ `?z:Tp` $\wedge$ `mary:(FM`$\sqcap\exists$`TC.AC)`

exam(paul,?y) ∧
subject(?y,?z) ∧ paul:St ∧ ?y:Co ∧ ?z:Tp ∧
expert(mary,?z) ∧ mary:(FM ⊓∃ TC.AC)

subject(ai,?z) ∧
paul:St ∧ ?y:Co ∧ ?z:Tp ∧
expert(mary,?z) ∧ mary:(FM ⊓∃ TC.AC)

paul:St ∧ ai:Co ∧ lp:Tp ∧
expert(mary,lp) ∧ mary:(FM⊓∃ TC.AC)

paul:St ∧ ai:Co ∧ lp:Tp ∧ mary:(FM⊓∃ TC.AC)

2. mayDoThesis(paul,mary)
   paul:St ∧ mary:NFP

Thus the DL-constraints associated with the derivations above are
paul:St(paul) ∧ ai:Co ∧ lp:Tp ∧ mary:(FM⊓∃ TC.AC)
and paul:St ∧ mary:NFP

A DL reasoner would be able to prove that in every model of the DL component of the knowledge base at least one of these constraints is true. The knowledge base includes the explicit assertions lp:Tp and paul:St. It remains to reason about mary:(FM⊓∃ TC. AC) and mary:NFP. The axiom about Mary says that she is a full professor and that all courses she teaches are advanced. She may or may not teach courses. In the first case as a full professor, she is a faculty member so that the constraint of the first derivation above is true. In the second case she is a non-teaching full professor and the constraint of the second derivation is true.

6.2.2 CARIN

CARIN [LR98] is defined as a family of languages with the intention to provide a hybrid integration of Datalog with different description logics. The class of DL logics considered includes any subset of the description logic $\mathcal{ALCNR}$, which admits the following class constructors (according to the terminology of Figure 4): intersectionOf, unionOf, complementOf, someValuesFrom, allValuesFrom, minCardinality, MaxCardinality and the intersection constructor for properties. CARIN allows unrestricted use of (ground or open) class-instance membership queries and property instance membership queries in rule bodies. The variables of the queries in the body of a CARIN rule need not appear elsewhere in the rule. Thus CARIN extends $\mathcal{AL}$-log by integrating Datalog with more expressive DL and by admitting more general DL queries in the bodies of the hybrid rules. The price of this generality is undecidability of query answering. However, as pointed out in [LR98], a refutation-complete query answering procedure can be obtained by combining SLD-resolution with the existential entailment algorithm for $\mathcal{ALCNR}$ described therein. Such a procedure terminates if there is an answer for a given query but may not terminate otherwise.

Decidable subsets of CARIN can be obtained, among others, by restriction to non-recursive rules or by restriction on the way in which variables can appear in the rules in property instance DL queries. The latter restriction requires that if a body of a rule includes a DL query of the form $p(x, y)$, where x

and y are variables, at least one of them appears in a non-DL body atom of this rule, whose predicate may only appear in facts or in body atoms, but not in rule heads. For both restricted subsets of CARIN [LR98] describes a sound and complete reasoning algorithm working in two steps: the DL-reasoning step and the rule reasoning step.

In the DL reasoning step, the DL-component of a given knowledge base is used to construct a set of its *completions*, each of which is represented by a finite set of DL-atoms and determines a *canonical model* of a program. A rule component of a CARIN knowledge base Δ, consisting of all hybrid rules and facts can now be augmented by the set of DL-assertions determined by a completion of the DL component. There is a finite number of such augmented rule components. In the rule reasoning step a standard forward chaining is done for each augmented rule component, using the added DL-assertions as new facts. A non-DL atom is entailed by the knowledge base iff it is entailed by each of its augmented rule components.

We illustrate CARIN by a simple example from [LR98]. Its DL component describes a classification of companies and related properties, while the hybrid rules describe the situation when a product can obtain a high price in a given country.

The DL component of the example hybrid knowledge base includes

- the class names:
  `european, american, associate, european-associate, american-associate, international, no-fellow-company`.
  Intuitively, instances of these classes are companies.

- the axioms

  $\texttt{european} \sqcap \texttt{american} \sqsubseteq \bot$
  $\texttt{european-associate} \equiv \exists \texttt{associate}.\ \texttt{european}$
  $\texttt{american-associate} \equiv \exists \texttt{associate}.\ \texttt{american}$
  $\texttt{international} \equiv \texttt{european-associate} \sqcup \texttt{american-associate}$
  $\texttt{no-fellow-company} \equiv \forall \texttt{associate}.\neg\, \texttt{american}$

  `b:international`

The rule component of the example hybrid knowledge base includes

- the predicates:
  `made-by(?x,?y)` *product* `?x` *is produced by company* `?y`,
  `monopoly(?x,?y,?z)` *company* `?x` *has monopoly for product* `?y` *in country* `?z`,
  `price(?x,?y,?z)` *product* `?x` *has in the country* `?y` *a price of level* `?z`

- the rules:
  $\texttt{price(?x,usa,high)} \leftarrow \texttt{made-by(?x,?y)} \wedge \texttt{?y:no-fellow-company}$
  $\texttt{price(?x,usa,high)} \leftarrow \texttt{made-by(?x,?y)} \wedge \texttt{monopoly(?y,?x,usa)}$
  $\wedge\ \langle\ \texttt{?y,?z}\rangle\texttt{:associate} \wedge \texttt{?z:american}$

- the facts:
  `made-by(a,b), monopoly(b,a,usa)`

We now discuss how the query `price(a,usa,high)` is answered by the method discussed in [LR98]. First, using the DL reasoner the canonical models of the DL-component are constructed. The DL component includes the assertion stating that `b` is an international company. As explained

in [LR98], in different canonical models constructed by the DL reasoner, b may or may not have american associates. In the first case the generated DL-atoms include the atoms $\langle$b, v $\rangle$:associate, v:american sufficient to derive the query atom by the second rule of the example. Otherwise, the rule component will be augmented by DL-atoms including the atom b:no-fellow-company and the query can be derived by the first rule.

Thus the example knowledge base entails the query.

6.2.3 Integrating Answer Set Programming with DL

Proposals to integrate function-free rules with answer set semantics and Description Logics are presented by several authors. The paper [Ros99] uses DL-queries of the $\mathcal{ALC}$ description logic as constraints in rule bodies. The rules allow disjunctive heads. The aspect of integration of separated DL-reasoner with a rule reasoner is not stressed in this approach. Yet another approach can be found in [HNV04, HV03b, HV03a]. In particular, the work [HNV04] presents a semantics with infinite open domains (see also [BDS97]) which is capable of dealing with an interesting fragment of OWL DL extended with rules. Again, this is a homogeneous system.

In this section we survey in more detail the paper [ELST04a] presenting a hybrid integration of OWL DL (and OWL Lite) (or more precisely the DL $\mathcal{SHOIN}(\mathbf{D})$ (and $\mathcal{SHIF}(\mathbf{D})$) with normal rules under answer set semantics [GL91]. Recall that normal rules may include two kinds of negation. So for example, if a research paper p in a cluster of papers is to be assigned to exactly one from among candidate reviewers, the following rules may be used to describe possible assignments:

$$\texttt{assign(?p,?r)} \leftarrow \texttt{candidate(?p,?r)} \wedge \sim\neg\texttt{assign(?p,?r)}$$
$$\neg\texttt{assign(?p,?s)} \leftarrow \texttt{candidate(?p,?s)} \wedge \texttt{assign(?p,?r)} \wedge \texttt{?r}\neq\texttt{?s}$$

Intuitively, the first of them says that a paper not known to be excluded from assignment to a candidate reviewer can be assigned to this reviewer. The second rule excludes new assignments for already assigned papers. The answer set semantics will then determine different assignments of reviewers by providing different answer sets.

A novelty of the approach is generalization of the form of DL-queries allowed in the bodies of the hybrid rules by so called dl-atoms. A general intuition of a generalized query is that it may refer to a variant of the DL-component obtained by modifying its assertions through adding extensions of the non-DL predicates. This means that DL-queries also allow for specifying an input from the rule component, and thus for a flow of information from the rule component to the DL-component, besides the flow vice versa, given by any DL-query. Hence, the approach allows for building rules on top of ontologies, but also (to some extent) building ontologies on top of rules. This is achieved by dynamic update operators through which the extensional part (i.e., class or property membership) of the DL-component can be modified for subjunctive querying. We illustrate this by an example based on a similar one in [ELST04a]. Assume that the DL-component has properties keyword and inArea, associating with research papers relevant keywords and areas. Assume that the rule component (the so-called *dl-program*) defines a predicate kw also associating papers with keywords, and a predicate paperArea associating papers with areas. The former may be defined by some facts and by a rule stating that if a paper has associated a keyword that also some other related keywords have to be associated with it.

$$\texttt{kw(?p,?w)} \leftarrow \texttt{kw(?p,?z)} \wedge \texttt{related(?z,?w)}$$

The latter may be defined by a hybrid rule containing a dl-atom which queries the property inArea in the DL-component augmented with additional assertions on the property keyword (which is known to influence inArea) defined by the given hybrid rules. Such a dl-atom is used as follows:

```
paperArea(?p,?a)  ←  DL[keyword∪kw; inArea](?p,?a)
```

The non-DL predicates specified in such a dl-atom are called input predicates. Conceptually, any type of update operator is possible. Apart from the above $\cup$ operation, which adds the extension of the non-DL predicate, further ones are available to increase the extension of the complement respectively constrain the extension of a class or property, thus facilitating the use of non-monotonic dl-atoms.

In this way, additional knowledge (gained in the program) can be supplied to the DL-component before querying. Using this mechanism, also more involved relationships between classes and/or properties can be defined and exploited. For the evaluation, the precise definition of queried DL classes and properties are fully transparent, and only the logical contents at the level of inference counts. Consequently, dl-programs fully support encapsulation and privacy of the DL-component — this is needed if parts of it should not be accessible (for example, if they contain an ontology about risk assessment in credit assignment), and only extensional reasoning services are available.

Two variants of formal semantics of such dl-programs — weak and strong answer set semantics — are defined in [ELST04a] as generalizations of the answer set semantics of normal programs. Additionally, [ELST04b] defines the well-founded semantics for dl-programs by suitably generalizing the notion of *unfounded sets* to the setting of dl-atoms. Alternatively, this semantics can be characterized in terms of the least and greatest fixpoint of a monotone operator similar as the well-founded semantics for ordinary normal programs. Analogously as for ordinary normal programs, the well-founded semantics for dl-programs approximates the strong answer set semantics for dl-programs. That is, every well-founded ground atom and no unfounded ground atom is a cautious (resp., brave) consequence of a dl-program under the strong answer set semantics.

As shown in [EISi05] a sound and complete reasoner for the hybrid language can be obtained by combining existing answer set reasoner for rules with an existing DL reasoner. This paper considers some technical issues regarding efficient implementation of both semantics, which has been carried out in a working prototype exploiting the two state-of-art tools DLV [LPF$^+$05] and RACER [HM01]. A major issue in this respect are the circular dependencies which may be created by input predicates, requiring interleaved calls of both reasoners. It is no longer possible to separate reasoning into two separate stages: the DL stage and the rule stage, as it was the case in $\mathcal{AL}$-log and CARIN. Special methods are devised for an efficient interfacing between the two reasoning systems at hand.

The distinguishing feature of this approach is the declarative problem solving paradigm of the answer set semantics. A problem is encoded to a non-monotonic logic program, such that its solutions can be extracted from corresponding answer sets of this program. Such an approach allows to resolve conflicts by virtue of permitting multiple intended models as alternative scenarios, which can prove useful for a range of applications with inherent nondeterminism, e.g., diagnosis and configuration using ontologies, or ontology merging.

7 Rules and Ontologies in F-logic

Michael Kifer

F-logic [KLW95] extends classical predicate calculus with the concepts of objects, classes, and types, which are adapted from object-oriented programming. In this way, F-logic integrates the paradigms of logic programming and deductive databases with the object-oriented programming paradigm.

Most of the applications of F-logic have been as a language for intelligent information systems based on the logic programming paradigm. This was the original motivation for the development of F-logic.

More recently, F-logic has been used to represent ontologies and other forms of Semantic Web reasoning [FES98, DBSA98, SM00, AL04, RLK04, KLPZ04].

Currently several implementations of the rule-based subset of F-logic are available. Ontobroker [Ont] is a commercial F-logic based engine developed by Ontoprise. It is designed as a knowledge-base component for a Java application. Flora-2 [YKZ02] is an open-source system that was developed at Stony Brook as part of a research project. Unlike Ontobroker which is designed to serve Java applications, Flora-2 is a complete programming environment for developing knowledge-intensive applications. It integrates F-logic with other novel formalisms such as HiLog [CKW93] and Transaction Logic [BK98]. TRIPLE [SDH02] is a partial implementation of F-logic with a particular emphasis on interoperability with RDF. Older, unmaintained F-logic based systems are also available, such as SILRI[20] and FLORID.[21]

In this section we first survey the main features of F-logic and then discuss its use as an ontology language.

7.1 Overview of F-logic

F-logic extends and subsumes predicate calculus both syntactically and semantically. In particular, it has a monotonic logical entailment relationship, and its proof theory is sound and complete with respect to the semantics. F-logic comes in two flavors: the first-order flavor and the logic programming flavor. The first-order flavor of F-logic can be viewed as a syntactic variant of classical logic, which makes an implementation through source-level translation possible [KLW95, YK00, YKZ02]. The logic programming flavor uses a subset of the syntax of F-logic, but gives it a different, non-first-order semantics.

To understand the relationship between the first-order variant of F-logic and its logic programming variant, recall that standard logic programming [Llo87b] is built on top of the rule-based subset of the classical predicate calculus by adding non-monotonic extensions. By analogy, object-oriented logic programming is constructed based on the rule-based subset of F-logic by adding the appropriate non-monotonic extensions [YKZ03, YKZ02, Ont]. These extensions are intended to capture the semantics of negation-as-failure, like in standard logic programming [VRS91], and the semantics of multiple inheritance with overriding (which does not arise in the standard case).

7.1.1 Basic Syntax

F-logic uses first-order variable-free terms to represent *object identity* (abbr., OID); for instance, john and father(mary) are possible Ids of objects. Objects can have single-valued or set-valued attributes. For instance,

mary[spouse $\rightarrow$ john, children $\twoheadrightarrow$ {alice,nancy}].
mary[children $\twoheadrightarrow$ {jack}].

Such formulas are called F-logic *molecules*. The first formula says that object mary has an attribute spouse, which is single-valued and whose value is the OID john. It also says that the attribute children is set-valued and its value is a set that *contains* two OIDs: alice and nancy. We emphasize "contains" because sets do not need to be specified all at once. For instance, the second formula above says that mary has an additional child, jack.

[20] http://ontobroker.semanticweb.org/silri/
[21] http://www.informatik.uni-freiburg.de/∼dbis/florid/

While some attributes of an object are specified explicitly, as facts, other attributes can be defined using deductive rules. For instance, we can derive john[children $\twoheadrightarrow$ {alice, nancy, jack}] using the following deductive rule:

X[children $\twoheadrightarrow$ {C}] :- Y[spouse $\rightarrow$ X, children $\twoheadrightarrow$ {C}].

Here we adopt the standard convention in logic programming that uppercase symbols denote variables while symbols beginning with a lowercase letter denote constants.

F-logic objects can also have *methods*, which are functions that take arguments. For instance,

john[grade(cs305,fall2004) $\rightarrow$ 100, courses(fall2004) $\twoheadrightarrow$ {cs305,cs306}].

says that john has a single-valued method, grade, whose value on the arguments cs305 (a course identifier) and fall2004 (a semester designation) is 100; it also has a set-valued method courses, whose value on the argument fall2004 is a set of OIDs that contains course identifiers cs305 and cs306. Like attributes, methods can be defined using deductive rules.

The F-logic syntax for *class membership* is john:student and for *subclass relationship* it is student::person. Classes are treated as objects and it is possible for the same object to play the role of a class in one formula and of an object in another. For instance, in the formula student:class, the symbol student plays the role of an object, while in student::person it appears in the role of a class.

In addition, F-logic provides the means for specifying schema information through *signature* formulas. For instance, person[name $\Rightarrow$ string, child $\Rightarrow\!\!>$ person] is a signature formula that says that class person has two attributes: a single-valued attribute name and a set-valued attribute child. It further says that, the first attribute returns objects of type string and the second returns sets of objects such that each object in the set is of type person. F-logic also supports first-order predicate syntax and in this way it extends classical predicate calculus and integrates the relational and object-oriented paradigms in knowledge representation.

We remark that attempts are being made to unify the syntax of the various implementations of F-logic, such as Ontobroker [Ont] and Flora-2 [YKZ02]. Among the more significant forthcoming changes (as far as this overview goes) are that all attributes will be treated as set-valued (for which $\rightarrow$ will be used instead of $\twoheadrightarrow$). To capture the single-valued attributes of old, cardinality constraints will be introduced. The syntax of variables will also change: instead of capitalization, all variables will be prefixed with the "?" sign.

7.1.2 Querying Meta-Information

F-logic provides simple and natural means for exploring the structure of object data. Both schema information associated with classes and the structure of individual objects can be queried by simply putting variables in the appropriate syntactic positions. For instance, to find the set-valued methods that are defined in the *schema* of class student and return objects of type person, one can ask the following query:

?- student[M $\Rightarrow\!\!>$ person].

The next query is about the type of the results of the attribute name in class student. In addition, the query returns all the superclasses of class student.

?- student::C and student[name $\Rightarrow$ T].

The above queries are *schema-level meta-queries* because they involve the subclass relationship and the type information (as indicated by the operators ::, $\Rightarrow$, and $\Rightarrow\!\!>$). In contrast, the following queries involve object data (rather than schema); they return the methods that have a known value for the object with the OID john:

?- john[SingleM $\rightarrow$ SomeValue].
?- john[SetM $\rightarrow\!\!>$ SomeValue].

Like the previous queries, the last two deal with meta-information about objects, but they examine object data rather than schema. Therefore, they are called *instance-level meta-queries*. The two kinds of meta-queries can return different results for several reasons. First, in case of semistructured data, schema information might be incomplete, so additional attributes might be defined for individual objects but not mentioned in the schema. Second, even if the schema is complete, the values of some attributes can be undefined for some objects. In this case, the undefined attributes will not be returned by instance-level meta-queries, but they would be returned by schema-level meta-queries.

7.1.3 Path Expressions

In addition to the basic syntax, F-logic supports so-called *path expressions*, which generalize the dot-notation in object-oriented programming languages such as Java or C++. Path expressions simplify navigation along attribute and method invocations, and help avoid explicit join conditions [FLU94].

A *single-valued* path expression, O.M, refers to the *unique* object R for which O[M $\rightarrow$ R] holds; a *set-valued* path expression, O..M, refers to some object, R, such that O[M $\rightarrow\!\!>$ {R}] holds. Here the symbols O and M can be either OIDs or other path expressions. Furthermore, M can be a method with arguments. For instance, O.M(P_1,. . .,P_k) is a valid path expression that refers to the object R that satisfies O[M(P_1,. . .,P_k) $\rightarrow$ R].

Since path expressions can occur anywhere an OID is allowed, they can be nested within other F-logic molecules and provide alternative and much more concise ways of addressing objects in a knowledge base. For instance, the path expression

Paper[authors $\rightarrow\!\!>$ {Author[name $\rightarrow$ john]}].publication..editors

refers to all editors of those papers in which john is the name of a coauthor. An equivalent representation in terms of the basic F-logic syntax would be

Paper[authors $\rightarrow\!\!>$ Author] and Author[name $\rightarrow$ john] and
Paper[publication $\rightarrow$ P] and P[editor $\rightarrow\!\!>$ E]

The reader has probably noticed the conceptual similarity between the path expressions in F-logic, introduced in [FLU94], and the language of XPath, which was developed later but with a similar purpose in mind.

7.1.4 Additional Features

F-logic includes a number of other language constructs that can be very useful in knowledge representation in general and on the Semantic Web in particular. One of these important features is the equality predicate, :=:, which can be used to declare two objects to be the same. For instance, mary :=: mother(john) asserts that the object with the OID mary and the object with the OID mother(john) are one and the same object. The presence of explicit equality goes against the grain of standard logic programming, which assumes a particular built-in theory of equality, where two variable-free terms are

equal if and only if they are identical. A common use of explicit equality on the Semantic Web is to provide assertions stating that a pair of syntactically different URIs refer to the same document.

Another important feature of some of the F-logic implementations, such as Flora-2, is integration with HiLog [CKW93]. This allows a higher degree of meta-programming in a clean and logical way. For instance, one can ask a query of the form

?- person[M(Arg) ⇒ person].

and obtain a set of all methods that take one argument, are declared to be part of the schema of class person, and return results that are objects belonging to class person. Note that M(Arg) is not a first-order term, since it has a variable in the position of a function symbol; such terms are not allowed in Prolog-based logic programming languages.

Later additions to F-logic include reification and anonymous object identity [YK03b, KLPZ04]. Both features are deemed to be important for Semantic Web and are included in RDF [KC03, Hay03]. It has been argued, however, that the RDF formalization of these notions is less than optimal and that the proposal requires significant extensions in order to be useful for advanced applications [YK03b]. A convincing use of the extensions provided by F-logic has been given in [KLPZ04] in the context of Semantic Web Services.

7.1.5 Inheritance

F-logic supports both *structural* and *behavioral* inheritance. The former refers to inheritance of method types from superclasses to their subclasses and the latter deals with inheritance of method definitions from superclasses to subclasses.

Structural inheritance is defined by very simple inference rules:

If subcl::cl, cl[attr $\star\Rightarrow$ type] then subcl[attr $\star\Rightarrow$ type]
If obj:cl, cl[attr $\star\Rightarrow$ type] then obj[attr $\Rightarrow$ type]

Similar rules hold for multi-valued attributes that are designated using the arrows $\star\Rightarrow\!\!>$ and $\Rightarrow\!\!>$. The statement cl[attr $\star\Rightarrow$ type] in the above rules states that attr is an *inheritable* attribute, which means that both its type and value are inheritable by the subclasses and members of class cl. Inheritability of the type of an attribute is indicated with the star attached to the symbol $\star\Rightarrow$. In all previous examples we have been dealing with *non-inheritable* attributes, which were designated with star-less arrows. Note that when the type of an attribute is inherited to a subclass it remains inheritable. However, when it is inherited to a member of the class it is no longer inheritable.

Type inheritance is not overridable; instead all types accumulate. For instance, from

faculty::employee.
manager::employee.
john:faculty.
faculty[reportsTo $\star\Rightarrow$ faculty].
employee[reportsTo $\star\Rightarrow$ manager].

we can derive two statements by inheritance: john[reportsTo $\star\Rightarrow$ faculty] and john[reportsTo $\star\Rightarrow$ manager]. The type expression for the more specific superclass, faculty, does not override the type expression for the less specific class, employee. The intended interpretation is that whoever john reports to must be both a manager and a faculty. These two statements can be replaced with a single statement of the form john[reportsTo $\star\Rightarrow$ (faculty and manager)].

Behavioral inheritance is more complex. To get a flavour of behavioral inheritance, consider the following small knowledge base:

royalElephant::elephant.
clyde:royalElephant.
elephant[color $\star\!\!\rightarrow$ grey].
royalElephant[color $\star\!\!\rightarrow$ white].

Like with type definitions, a star attached to the arrow, $\star\!\!\rightarrow$, designates an inheritable method. For instance, color is an inheritable attribute in classes elephant and royalElephant. The inference rule that guides behavioral inheritance can informally be stated as follows. If obj is an object and cl is a class, then

obj:cl, cl[attr $\star\!\!\rightarrow$ value] should imply obj[attr $\rightarrow$ value]

unless the inheritance is overwritten by a more specific class. The meaning of the exception here is that the knowledge base should not imply the formula obj[attr $\rightarrow$ value] if there is an intermediate class, cl', which overrides the inheritance, i.e., if obj:cl', cl'::cl are true and cl'[attr $\star\!\!\rightarrow$ value'] is defined explicitly.[22] A similar exception exists in case of multiple inheritance conflicts. Note that inheritable attributes become non-inheritable after they are inherited by class members. In the above case, inheritance of the grey color is overwritten by the white color and so clyde[color $\rightarrow$ white] is derived by the rule of inheritance.

7.1.6 Semantics

The semantics of F-logic is based on the notion of F-structures, which extend the notion of semantic structures in classical predicate calculus. OIDs are interpreted in F-structures as elements of the domain and methods (and attributes) are interpreted as partial functions of suitable arities. The first argument of each such function is the Id of the object in whose context the method or the attribute is defined. Signature formulas are interpreted by functions whose properties are made to fit the common properties of types. Details of F-structures can be found in [KLW95].

Armed with the notion of the F-structures, a first-order entailment relation is defined in a standard way: $\phi \models \psi$ if and only if every F-structure that satisfies ϕ also satisfies ψ. This entailment together with the sound and complete resolution-based proof theory [KLW95] are the basis of the first-order variant of F-logic.

The semantics of the logic programming variant of F-logic is built by analogy with the corresponding development in deductive databases. The meaning of negation is made non-monotonic and is based on an extension of the well-founded semantics [VRS91]. The interesting and nontrivial aspect of this extension is not due to negation (negation is handled analogously to [VRS91]) but due to behavioral inheritance with overriding. Earlier we have seen an informal account of inference by inheritance. Although the rules of such inference seem natural, they present subtle problems when behavioral inheritance is used together with deductive rules. To understand the problem, consider the following example.

cl[attr $\star\!\!\rightarrow$ v1].
subcl::cl.
obj:subcl.
subcl[attr $\star\!\!\rightarrow$ v2] :- obj[attr $\rightarrow$ v1].

If we apply the rule of inheritance to this knowledge base, then obj[attr $\rightarrow$ v1] should be inherited, since no overriding takes place. However, once obj[attr $\rightarrow$ v1] is derived by inheritance, subcl[attr $\star\!\!\rightarrow$ v2] can be derived by deduction, and now we have a chicken-and-egg problem. Since subcl is a more

[22] The notion of an explicit definition seems obvious at first but, in fact, is quite subtle [YK02].

specific superclass of obj, the derivation of subcl[attr $\star\!\!\rightarrow$ v2] appears to override the earlier inheritance of obj[attr $\rightarrow$ v1]. But this, in turn, undermines the reason for deriving subcl[attr $\star\!\!\rightarrow$ v2]. The above is only one of several suspicious derivation patterns that arise due to interaction of inheritance and deduction. The original solution reported in [KLW95] was not model-theoretic and was problematic in several other respects as well. A satisfactory and completely model-theoretic solution was proposed in [YK02, YK03a].

7.2 F-logic as an Ontology Language

From the beginning, F-logic has been viewed as a natural candidate for an ontology language due to its direct support for object-oriented concepts, its frame-based syntax, and extensive support for meta-programming [FES98, DBSA98, SM00]. More recently it has been adopted as a basis for ontology languages for Semantic Web Services [RLK04, BGG$^+$05].

7.2.1 The Basic Techniques

A typical ontology includes three main components:

1. *A taxonomy of classes*. This includes the specification of the class hierarchy, i.e., which classes are subclasses of other classes.

2. *Definitions of concepts*. These definitions specify the allowed attributes of each class, their types, and other properties (like symmetry or transitivity).

3. *Definitions of instances*. Instances (i.e., concrete data objects) are defined by indicating which concepts (i.e., classes) they belong to and by specifying concrete values for the attributes of those instances. Sometimes the values might not be given explicitly, but only their existence can be asserted with various degrees of precision. For instance, $\exists$ F john[father $\rightarrow$ F] or john[father $\rightarrow$ bob] $\vee$ john[father $\rightarrow$ bill]. Some concepts may not have explicitly defined instances. Instead, their instances may be defined by deductive rules. Such concepts are akin to database views.

In F-logic, class taxonomies are represented directly using the subclass relationship ::. Concept definitions are represented using signature formulas, such as person[name $\star\!\!\Rightarrow$ string, spouse $\star\!\!\Rightarrow$ person]. Special properties of certain attributes can be expressed using rules. For instance, to state that spouse is a symmetric relationship in class person one can write

X[spouse $\rightarrow$ Y] :- Y:person and Y[spouse $\rightarrow$ X].

Finally, instance definitions can be specified as facts using data molecules as follows:

john:student.
john[name $\rightarrow$ John, address $\rightarrow$ '123 Main St.', spouse $\rightarrow$ Mary].

Derived classes can be defined using rules. For instance, if the concepts of student and employee are already defined, we can define a new concept, workstudy using the following statements:

X:workstudy :- X:(student and employee) and X[jobtype $\rightarrow$ J] and J:clerical.

Properties can also be defined using rules. For instance, if the properties mother and father are already defined, we can define the properties of parent and ancestor as follows:

X[parent → P] :− X[mother → P].
X[parent → P] :− X[father → P].
X[ancestor → A] :− X[parent → A].
X[ancestor → A] :− X[parent → P] and P[ancestor → A].

Various implementations of F-logic introduced several forms of more concise syntax. For instance, the workstudy rule above can be written as

X:workstudy :− X[jobtype → J:clerical]:(student and employee).

the two parent rules can be abbreviated to

X[parent → P] :− X[mother → P or father → P]].

and the second ancestor rule can be written as

X[ancestor → A] :− X[parent → P[ancestor → A]].

7.2.2 Relationship to Description Logics

No discussion of F-logic is complete without a comparison with description logics (abbr. DL) [BCM+03] and, in particular, with languages such as OWL [PSHH03]. Since the first-order flavor of F-logic is an extension of classical predicate logic, it is clear that a description logic subset can be defined within F-logic and, indeed, this has been done [Bal95]. In this sense, F-logic subsumes DLs. However, as mentioned earlier, most applications of F-logic (and all implementations known to us) use the logic programming flavor of the logic so a proper comparison would be made with that flavor.

Unlike DLs, F-logic is computationally complete. This can be a blessing or a curse depending on how one looks at this matter. On one hand, the expressive power of F-logic provides for a simple and clear specification of many problems that are beyond the expressive power of any DL. On the other hand, expressive F-logic knowledge bases provide no computational guarantees. However, many workers in the field dismiss this problem as a non-issue for two reasons:

- The exponential complexity of many problems in description logics provides very little comfort in practice, especially in reasoning with large ontologies.

- A vast class of computational problems in F-logic is decidable and has polynomial complexity. This includes all queries to knowledge bases that do not use function symbols and includes a large subclass of queries that are beyond the expressive power of DLs. Furthermore, research in logic programming and deductive database has identified large classes of knowledge bases *with* function symbols where query answering is decidable (for instance, [NS97]).

Nevertheless, there are two aspects where DLs provide more flexibility. First, DLs allow the user to represent existential information. For instance, one can say that there is a person with certain properties without specifying any concrete instance of such a person. In F-logic one can express only an approximation of such a statement using Skolem functions. Similarly, DLs admit disjunctive information into the knowledge base. For instance, one can say that John has a book or a bicycle. The corresponding statement in F-logic is only an approximation:

john[has → _#:(book or bicycle)].

The symbol _# here denotes a unique Skolem constant that does not occur anywhere else in the knowledge base. While this may be an acceptable approximation in some cases, it is still significantly weaker that the corresponding DL statement.

For instance, if upon closer examination it becomes known that John does not have a book, then in DLs we would conclude that John has a bicycle. In the logic programming flavor of F-logic (as in other logic programming systems) we cannot even state that John has no books directly—one has to employ some rather complex tricks. Some extensions of standard logic programming support *explicit negation* and thus can make negative information easier to specify. For instance, this problem could be overcome by combining F-logic with Courteous Logic Programming [Gro97, Gro99]. Other extensions allow disjunctive information in the rule heads [Prz94, LMR92], which permits statements like $\texttt{john}[\texttt{father} \rightarrow \texttt{bob}] \vee \texttt{john}[\texttt{father} \rightarrow \texttt{bill}]$.

7.2.3 Example: An OWL-S Profile

We now give a more extensive example of an ontology specified using F-logic—part of an OWL-S profile [Coa04]. OWL-S is an OWL-based Web ontology, which is intended to provide Web service providers with a core set of constructs for describing the properties and capabilities of their Web services. OWL-S often refers to externally defined data types using the namespace notation. Although some implementation of F-logic support URIs and namespaces, our example will omit all namespace definitions and will reference the corresponding external data types and concepts by enclosing them in single quotes, e.g., `'xsd:string'`.

```
'service:ServiceProfile' : 'owl:Class'.
'Profile' :: 'service:ServiceProfile'
'Profile'[
   serviceName *=> 'xsd:string',
   textDescription *=> 'xsd:string',
   'rdfs:comment'*->'Definition of Profile',
   contactInformation *=>> 'Actor',
   hasProcess *=> 'process:Process',
   serviceCategory *=>> ServiceCategory,
   serviceParameter *=>> ServiceParameter,
   hasParameter *=>> 'process:Parameter',
   hasInput *=>> 'process:Input',
   hasOutput *=>> 'process:ConditionalOutput',
   hasPrecondition *=>> 'expr:Condition',
   hasEffect *=>> 'process:ConditionalEffect'
].

hasInput[subpropertyof ->> hasParameter].
hasOutput[subPropertyOf ->> hasParameter].

// Definition of subPropertyOf
Obj[P ->> Val] :- S[subPropertyOf ->> P] and Obj[S ->> Val].

'ServiceCategory' : 'owl:Class'.
'ServiceCategory'[
   categoryName *=> 'xsd:string',
```

```
   taxonomy *=> 'xsd:string',
   value *=> 'xsd:string',
   code *=> 'xsd:string'
].

'ServiceParameter' : 'owl:Class'.
'ServiceParameter'[
    serviceParameterName *=> 'xsd:string',
    sParameter *=> 'owl:Thing'
].

'Actor' : 'owl:Class'.
'process:Process' : 'owl:Class'.
'expr:Condition' : 'owl:Class'.
'process:Input' : 'owl:Class'.
'process:ConditionalOutput' : 'owl:Class'.
'process:ConditionalEffect' : 'owl:Class'.
'process:Parameter' : 'owl:Class'.
```

The above ontology is fairly simple. The frame-based syntax of F-logic enables concise and clear description of the properties of the various classes defined by OWL-S. The only place where a more sophisticated aspect of F-logic is necessary is the definition of `subPropertyOf`, a property that applies to attributes when they are considered as objects in their own right. To define the meaning of this property we use an F-logic rule.

OWL distinguishes between *object properties* and *data type properties*, and defines two OWL classes for that. The class 'owl:ObjectProperty' is populated by object properties, which are attributes whose range is an OWL class. The class 'owl:DataTypeProperty' is populated by data type properties, which are defined as attributes whose range is an XML type, such as 'xsd:string'. In the OWL-based OWL-S ontology, every property must be explicitly specified to be in either the 'owl:ObjectProperty' class or the 'owl:DataTypeProperty' class. In F-logic this can be done much more elegantly using rules:

```
Prop : property(Range) :- Domain[Prop *=> Range or Prop *=>> Range].
Prop : 'owl:ObjectProperty' :-
            Prop : property(Range) and Range : 'owl:Class'.
Prop : 'owl:DataTypeProperty' :-
            Prop : property(Range) and not Range : 'owl:Class'.
```

This example provides a glimpse on how the ability of F-logic to operate at the meta-level provides significant benefits in terms of conciseness and readability of ontology specifications.

8 Summary and Discussion

This document surveys proposals for extending the Semantic Web ontology layer with rules. We are not aware of any other complete survey of these proposals. The ongoing REWERSE work aims at definition and deployment of a collection of inter-operable reasoning languages for advanced Web systems. This state-of-the-art survey provides a perspective of the related efforts and is thus essential for the REWERSE work.

We focused on the work referring to the existing W3C Recommendations RDF (and RDFS) and OWL. The motivation for such an extension is the restricted expressive power of RDF and OWL, and the fact that in many applications rules appear as a natural KR paradigm.

The integration of the standard Resource Description Framework with rules does not bring any significant new theoretical problems. As discussed in Section 2.1, it has a first-order based model theory that can be almost immediately implemented in existing rule-based languages, in particular logic programming languages with tabulation based proof procedures. Inference in RDF(S) is decidable and can be encoded into first-order Horn theories with only binary predicates. The problems are more of technological character, in particular how to deal efficiently with immense sizes of triple bases. Existing work on integration of RDF and rules focuses on building prototype systems, without paying too much attention to formal semantics and complexity analysis. Currently, there is no implementation supporting the collection of features that can be found in different existing prototypes, discussed in Section 3. This is partly due to the lack of standardization, but the RuleML initiative aims at creating such standards.

The Web Ontology Language OWL is based on a Description Logic, and the question how to combine it with rules can be studied in the context of a general problem how to integrate Description Logics with rule languages based on logic. A common framework for addressing this problem has been proposed in [FT04]. This paper defines formally three approaches to this general problem, and classifies accordingly the existing proposals for integration. Two of the defined approaches correspond essentially to what we call, respectively, the homogeneous approach and the hybrid approach. The third one, the autoepistemic approach described in [DLN$^+$98, DNR02], is, to our knowledge, not yet represented in the work addressing integration of RDF or OWL with rules, and is therefore not discussed in this paper.

We surveyed the proposals addressing integration of OWL with rules. The DLP approach, discussed in Section 4 defines an intersection of the Description Logic underlying OWL and Horn clauses, thus making possible re-use of existing reasoners. The resulting logic is decidable. Due to its nature, the DLP does not increase the expressive power of OWL, which is the main objective for introducing rules. On the other hand, SWRL (Section 5), the emerging extension of OWL with rules results in an undecidable logic. It has a well-defined declarative semantics, based on FOL, but the issue of SWRL reasoning is still a subject of research. The paper [MSS04a] identifies a decidable subset of SWRL and shows how it can be compiled into disjunctive logic programs, thus opening for re-use of reasoning algorithms of disjunctive LP for SWRL reasoning. Ongoing discussion on further extensions of SWRL towards full FOL may result in new proposals.

Implementation of DL reasoning in logic and constraint programming, discussed previously by several authors, (e.g. [FH95, MV03, HMS04, Swi04, MSS04b, HNV04]) may be considered a natural basis for integration of rules and ontologies. A logic programming variant of F-logic, surveyed in Section 7 allows for embedding rules and DL ontologies. It also supports non-monotonic extensions capturing the semantics of negation-as-failure and multiple inheritance with overriding.

The above mentioned proposals aim at creation and use of expressive logical languages for the Semantic Web. Such languages are to be supported by dedicated reasoners, thus the proposals follow the homogeneous approach. While creating such languages seems to be very desirable, we note that separate reasoners for OWL and for rule languages are already available on-line and new ones will certainly emerge. Moreover rule applications may use different specialized rule languages, supported by specialized reasoners (for example, certain applications may require non-monotonic reasoning, while for others a monotonic logic is sufficient). In these cases a hybrid integration of existing rule reasoners with existing ontology reasoners may provide a convenient alternative to the use of a general reasoner. Some proposal for hybrid integration of rules and ontologies were discussed in Section 6. In particular the work discussed in Section 6.2.3, shows that hybrid integration need not be restricted to placing rules on top of ontologies, and shows a possibility of passing information from rules to ontologies. From

the perspective of REWERSE, where a collection of inter-operable languages is being developed, the hybrid approach aiming at composition and re-use of existing reasoners may facilitate achievement of the objectives.

We now briefly comment on existing and planned REWERSE activities related to the work surveyed in this paper.

Integration of non-monotonic constructs in RDF(S) rules

The RDF community has been reluctant to accept general non-monotonic reasoning in the Semantic Web; according to the opposers, this might introduce some brittleness in reasoning. However, it has been recognized that some "safe" forms of (local) closed world assumptions are necessary for real-world applications. In particular Policy Specifications which are in the focus of REWERSE WG I2 may require non-monotonic reasoning. This is being actively studied by REWERSE Working Group I1 (see [AADW04]). One possible way of compromise, is to allow two forms of negation, open and closed, to specify negative knowledge in the Semantic Web. The necessary theory already exists, but still some work is necessary to adapt it to the Semantic Web setting (see [Dam05a] for a list of problems and some possible solutions).

Contributions to Hybrid Integration

The REWERSE Working Group I3 develops composition models and composition technologies for Semantic Web languages, following the invasive software composition approach [Aßm03]. Developing a framework for hybrid composition of RuleML rules and OWL ontologies fits well in these objectives. An example application under development is a reasoner for a subset of SWRL, implemented by hybrid integration of an existing OWL reasoner with a Datalog reasoner. In the long range the objective is to assure interoperability of the emerging REWERSE languages.

Extending Xcerpt and XChange to Semantic Web languages

Xcerpt[23] is a deductive rule-based query language for data on the Web, developed by REWERSE WG I4. Originally focused on XML it is presently being extended to other layers of the Semantic Web. Thus Xcerpt reasoner is also to handle ontology reasoning in the presence of RDF or OWL data. It may be interesting to investigate if the hybrid approach can be applied in that case, for interfacing the existing ontology reasoners with basic Xcerpt reasoner. XChange is a high-level language for programming reactive behaviour on the Semantic Web. It builds upon Xcerpt and is developed in REWERSE. Thus, hybrid techniques developed for Xcerpt should also be applicable to XChange.

References

[AA02] G. Antoniou and M. Arief. Executable declarative business rules and their use in electronic commerce. In *Proc. of ACM Symposium On Applied Computing*, 2002.

[AADW04] Anastasia Analyti, Grigoris Antoniou, Carlos Viegas Damásio, and Gerd Wagner. Negation and negative information in the w3c resource description framework. *Annals of Mathematics, Computing & Teleinformatics*, 2(1):25–34, 2004.

[23] http://www.xcerpt.org

[ABGM00] G. Antoniou, D. Billington, G. Governatori, and M.J. Maher. A flexible framework for defeasible logics. In *Proc. AAAI 2000*, pages 405–410, 2000.

[ABGM01] G. Antoniou, D. Billington, G. Governatori, and M.J. Maher. Representation results for defeasible logic. *ACM Transactions on Computational Logic*, 2(2):255–287, 2001.

[ABM99] G. Antoniou, D. Billington, and M.J. Maher. On the analysis of regulations using defeasible rules. In *Proc. 32nd Hawaii International Conference on Systems Science*, 1999.

[ABW04] G. Antoniou, A. Bikakis, and G. Wagner. A flexible framework for defeasible logics. In *Proc. RuleML'04*, LNCS 3323. Springer-Verlag, 2004.

[AL04] J. Angele and G. Lausen. Ontologies in F-logic. In S. Staab and R. Studer, editors, *Handbook on Ontologies in Information Systems*, pages 29–50. Springer Verlag, Berlin, Germany, 2004.

[AMB00] G. Antoniou, M.J. Maher, and D. Billington. Defeasible logic versus logic programming. *Journal of Logic Programming*, 41(1):45–57, 2000.

[APM+04] R. Ashri, T. Payne, D. Marvin, M. Surridge, and S. Taylor. Towards a semantic web security infrastructure. In *Proc. of Semantic Web Services 2004 Spring Symposium Series*. Stanford University, 2004.

[Aßm03] Uwe Aßmann. *Invasive Software Composition*. Springer-Verlag, February 2003.

[BAK91] Roland N. Bol, Krzysztof R. Apt, and Jan Willem Klop. An analysis of loop checking mechanisms for logic programs. *Theoretical Computer Science*, 86(1):35–79, August 1991.

[Bal95] Mira Balaban. The f-logic approach for description languages. *Annals of Mathematics and Artificial Intelligence*, 15(1):19–60, 1995.

[BAV04] Nick Basiliades, Grigoris Antoniou, and Ioannis Vlahavas. DR-DEVICE: A Defeasible Logic System for the Semantic Web. In *Principles and Practice of the Semantic Web Reasoning*, volume 3208 of *LNCS*, pages 134–148. Springer-Verlag, 2004.

[BCM+03] Franz Baader, Diego Calvanese, Deborah McGuinness, Daniele Nardi, and Peter Patel-Schneider, editors. *The Description Logic Handbook*. Cambridge University Press, 2003.

[BDS97] Kristof Van Belleghem, Marc Denecker, and Danny De Schreye. A strong correspondence between description logics and open logic programming. In *ICLP*, pages 346–360, 1997.

[Bec04] Dave Beckett. RDF/XML Syntax Specification (Revised). W3C Recommendation, 10 February 2004. Available at `http://www.w3.org/TR/2004/REC-rdf-syntax-grammar-20040210/`.

[Ber66] R. Berger. The undecidability of the dominoe problem. *Mem. Amer. Math. Soc.*, 66:1–72, 1966.

[BG94] Chitta Baral and Michael Gelfond. Logic programming and knowledge representation. *Journal of Logic Programming*, 19/20:73–148, 1994.

[BG00] Dan Brinkley and R. V. Guha. Resource description framework (RDF) schema specification 1.0. W3C Candidate Recommentation, March 2000.
http://www.w3.org/TR/2000/CR-rdf-schema-20000327/.

[BGG+05] D. Berardi, B. Grosof, M. Gruninger, R. Hull, M. Kifer, D. Martin, S. McIlraith, and J. Su. Semantic web services language. Technical report, Semantic Web Services Initiative, February 2005. http://www.daml.org/services/swsl/, in preparation.

[BK98] A.J. Bonner and M. Kifer. A logic for programming database transactions. In J. Chomicki and G. Saake, editors, *Logics for Databases and Information Systems*, chapter 5, pages 117–166. Kluwer Academic Publishers, March 1998.

[BL98] Tim Berners-Lee. Notation 3. W3C Recommendation, 1998. Available at http://www.w3.org/DesignIssues/Notation3.html.

[BLHL01] Tim Berners-Lee, James Hendler, and Ora Lassila. The semantic web. *Scientific American*, 2001.

[Bor96] Alexander Borgida. On the relative expressiveness of description logics and predicate logics. *Artificial Intelligence*, 82(1–2):353–367, 1996.

[BPS94] Alexander Borgida and Peter F. Patel-Schneider. A semantics and complete algorithm for subsumption in the CLASSIC description logic. *J. of Artificial Intelligence Research*, 1:277–308, 1994.

[BS96] Franz Baader and Ulrike Sattler. Number restrictions on complex roles in description logics: A preliminary report. In *Proc. of the 5th Int. Conf. on the Principles of Knowledge Representation and Reasoning (KR'96)*, pages 328–338, 1996.

[BvHH+04] Sean Bechhofer, Frank van Harmelen, Jim Hendler, Ian Horrocks, Deborah L. McGuinness, Peter F. Patel-Schneider, and Lynn Andrea Stein. OWL Web Ontology Language Reference. W3C Recommendation, 10 February 2004. Available at http://www.w3.org/TR/2004/REC-owl-ref-20040210/.

[CDGL98] Diego Calvanese, Giuseppe De Giacomo, and Maurizio Lenzerini. On the decidability of query containment under constraints. In *Proc. of PODS'98*, pages 149–158, 1998.

[CKW93] W. Chen, M. Kifer, and D.S. Warren. HiLog: A foundation for higher-order logic programming. *Journal of Logic Programming*, 15(3):187–230, February 1993.

[Coa04] OWL-S Coalition. Owl-s. http://www.daml.org/services/owl-s/1.1/, December 2004.

[CWM] CWM - Closed World Machine. Available at http://www.w3.org/2000/10/swap/doc/cwm.html.

[Dam05a] Carlos Viegas Damásio. Open and closed reasoning in the semantic web. Technical report, Centro de Inteligência Artificial da Universidade Nova de Lisboa, 2005. Available from http://centria.di.fct.unl.pt/ cd/publicacoes/openclosed.pdf.

[Dam05b] Carlos Viegas Damásio. SEW - A SEmantic Web engine, 2005. Available at http://centria.di.fct.unl.pt/~cd/projectos/w4.

[DBSA98] S. Decker, D. Brickley, J. Saarela, and J. Angele. A query and inference service for RDF. In *QL'98 - The Query Languages Workshop*, December 1998.

[DG84] W. Dowling and J. Gallier. Linear time algorithms for testing the satisfiability of propositional horn formulae. *Journal of Logic Programming*, 3:267–284, 1984.

[DLN+98] Francesco M. Donini, Maurizio Lenzerini, Daniele Nardi, Werner Nutt, and Andrea Schaerf. An epistemic operator for description logics. *Artificial Intelligence*, 100(1–2):225–274, 1998.

[DLNS98] Francesco M. Donini, Maurizio Lenzerini, Daniele Nardi, and Andrea Schaerf. AL-log: Integrating datalog and description logics. *Journal of Intelligent Information Systems*, 10(3):227–252, 1998.

[DNR02] Francesco M. Donini, Daniele Nardi, and Riccardo Rosati. Description logics of minimal knowledge and negation as failure. *ACM Trans. Comput. Log.*, 3(2):177–225, 2002.

[EISi05] Thomas Eiter, Giovambattista Ianni, Roman Schindlauer, and Hans Tomp its. Nonmonotonic Description Logic Programs: Implementation and Experiments . In F. Baader and A. Voronkov, editors, *Proceedings 12th International Conference on Logic for Programming, Artificial Intelligence and Reasoning (LPAR 2005)*, LNCS. Springer, 2005.

[ELST04a] Thomas Eiter, Thomas Lukasiewicz, Roman Schindlauer, and Hans Tompits. Combining Answer Set Programming with Description Logics for the Semantic Web". In Didier Dubois, Christopher Welty, and Mary-Anne Williams, editors, *Proceedings Ninth International Conference on Principles of Knowledge Representation and Reasoning (KR 2004), June 2-5, Whistler, British Columbia, Canada*, pages 141–151. Morgan Kaufmann, 2004.

[ELST04b] Thomas Eiter, Thomas Lukasiewicz, Roman Schindlauer, and Hans Tompits. Well-founded Semantics for Description Logic Programs in the Semantic Web. In *Proceedings RuleML 2004*, number 3323 in LNCS, pages 81–97. Springer, 2004.

[FES98] D. Fensel, M. Erdmann, and R. Studer. Ontobroker: How to make the WWW intelligent. In *Proceedings of the 11th Banff Knowledge Acquisition for Knowledge-Based Systems Workshop*, Banff, Canada, 1998.

[FH95] T. Fruhwirth and P. Hanschke. Terminological reasoning with constraint handling rules. In V. Saraswat and P. Van Hentenryck, editors, *Principles and Practice of Constraint Programming*. MIT Press, 1995.

[FLU94] J. Frohn, G. Lausen, and H. Uphoff. Access to objects by path expressions and rules. In *Int'l Conference on Very Large Data Bases*, pages 273–284, Santiago, Chile, 1994. Morgan Kaufmann, San Francisco, CA.

[FMG02] Aykut Firat, Stuart Madnick, and Benjamin N. Grosof. Financial information integration in the presence of equational ontological conflicts. In *Proc. of WITS-2002*, 2002.

[FMGar] Aykut Firat, Stuart Madnick, and Benjamin N. Grosof. Knowledge integration to overcome ontological heterogeneity: Challenges from financial information systems. In *Proc. of ICIS-2002*, To appear.

[FT04] Enrico Franconi and Sergio Tessaris. Rules and Queries with Ontologies: A Unified Logical Framework. In *Principles and Practice of the Semantic Web Reasoning*, volume 3208 of *LNCS*, pages 50–60. Springer-Verlag, 2004.

[GB04] Jan Grant and Dave Beckett. RDF Test Cases. W3C Recommendation, 10 February 2004. Available at `http://www.w3.org/TR/2004/REC-rdf-testcases-20040210/`.

[GDtHO01] G. Governatori, M. Dumas, A. ter Hofstede, and P. Oaks. A formal approach to legal negotiation. In *Proc. ICAIL*, pages 168–177, 2001.

[GGF02] B.N. Grosof, M.D. Gandhe, and T.W. Finin. Sweetjess: Translating damlruleml to jess. In *Proc. International Workshop on Rule Markup Languages for Business Rules on the Semantic Web*, 2002.

[GH03] R. V. Guha and P. Hayes. Lbase: Semantics for languages of the semantic web. W3C Note, 10 October 2003. Available at `http://www.w3.org/TR/2003/NOTE-lbase-20031010/`.

[GHVD03a] B. Grosof, I. Horrocks, R. Volz, and S. Decker. Description logic programs: Combining logic programs with description logic, 2003.

[GHVD03b] Benjamin N. Grosof, Ian Horrocks, Raphael Volz, and Stefan Decker. Description logic programs: Combining logic programs with description logic. In *Proc. of the Twelfth International World Wide Web Conference (WWW 2003)*, pages 48–57. ACM, 2003.

[GL88] M. Gelfond and V. Lifschitz. The stable model semantics for logic programming. In *Logic Programming, Proc. of the Fifth International Conference and Symposium*, pages 1070–1080. MIT Press, 1988.

[GL91] M. Gelfond and V. Lifschitz. Classical Negation in Logic Programs and Disjunctive Databases. *New Generation Computing*, 9:365–385, 1991.

[GLC99] Benjamin N. Grosof, Yannis Labrou, and Hoi Y. Chan. A declarative approach to business rules in contracts: Courteous logic programs in xml. In *Proc. of EC-99*, 1999.

[GP02] Benjamin N. Grosof and Terrence C. Poon. Representing agent contracts with exceptions using xml rules, ontologies, and process descriptions. In *Proc. of International Workshop on Rule Markup Languages for Business Rules on the Semantic Web*, 2002.

[Grä99] Erich Grädel. On the restraining power of guards. *J. of Symbolic Logic*, 64:1719–1742, 1999.

[Gro97] B.N. Grosof. Prioritized conflict handling for logic programs. In *International Logic Programming Symposium*, pages 197–211, 1997.

[Gro99] B.N. Grosof. A courteous compiler from generalized courteous logic programs to ordinary logic programs. Technical Report RC 21472, IBM, July 1999.

[Hay03] Patrick Hayes. RDF model theory. W3C Working Draft, 10 October 2003. Available at `http://www.w3.org/TR/rdf-mt/`.

[Hay04] Patrick Hayes. RDF Semantics. W3C Recommendation, 10 February 2004. Available at `http://www.w3.org/TR/2004/REC-rdf-mt-20040210/`.

[HEPS03] Masahiro Hori, Jérôme Euzenat, and Peter F. Patel-Schneider. OWL web ontology language XML presentation syntax. W3C Note, 11 June 2003. Available at `http://www.w3.org/TR/owl-xmlsyntax/`.

[HM01] V. Haarslev and R. Möller. RACER system description. In *Proceedings IJCAR-2001*, volume 2083 of *LNCS*, pages 701–705, 2001.

[HMS04] Ullrich Hustad, Boris Motik, and Ulrike Sattler. Reducing SHIQ-Description Logic to Disjunctive Daalog Programs. In *Principles of Knowledge Representation and Reasoning: Proc. of KR2004*, pages 152–162. AAAI Press, 2004.

[HNV04] Stijn Heymans, Davy Van Nieuwenborgh, and Dirk Vermeir. Semantic web reasoning with conceptual logic programs. In Grigoris Antoniou and Harold Boley, editors, *RuleML*, volume 3323 of *Lecture Notes in Computer Science*, pages 113–127. Springer, 2004.

[HPS04] Ian Horrocks and Peter F. Patel-Schneider. A proposal for an OWL rules language. In *Proc. of the Thirteenth International World Wide Web Conference (WWW 2004)*, pages 723–731. ACM, 2004.

[HPSB+04] Ian Horrocks, Peter F. Patel-Schneider, Harold Boley, Said Tabet, Benjamin Grosof, and Mike Dean. SWRL: A semantic web rule language combining owl and ruleml. W3C Note, 21 May 2004. Available at `http://www.w3.org/Submission/SWRL/`.

[HS03] Ian Horrocks and Ulrike Sattler. The effect of adding complex role inclusion axioms in description logics. In *Proc. of the 18th Int. Joint Conf. on Artificial Intelligence (IJCAI 2003)*, pages 343–348. Morgan Kaufmann, Los Altos, 2003.

[HST99] Ian Horrocks, Ulrike Sattler, and Stephan Tobies. Practical reasoning for expressive description logics. In *Proc. of the 6th Int. Conf. on Logic for Programming and Automated Reasoning (LPAR'99)*, pages 161–180, 1999.

[HSTT00] I. Horrocks, U. Sattler, S. Tessaris, and S. Tobies. How to decide query containment under constraints using a description logic. In *Proc. of LPAR'2000*, 2000.

[HV03a] Stijn Heymans and Dirk Vermeir. Integrating description logics and answer set programming. In François Bry, Nicola Henze, and Jan Maluszynski, editors, *PPSWR*, volume 2901 of *Lecture Notes in Computer Science*, pages 146–159. Springer, 2003.

[HV03b] Stijn Heymans and Dirk Vermeir. Integrating semantic web reasoning and answer set programming. In Marina De Vos and Alessandro Provetti, editors, *Answer Set Programming*, volume 78 of *CEUR Workshop Proceedings*, 2003.

[Jen] Jena: A Semantic Web Framework for Java . Available at `http://jena.sourceforge.net/index.html`.

[KC03] Graham Klyne and Jeremy J. Carroll. Resource description framework (RDF): Concepts and abstract syntax. W3C Working Draft, 10 October 2003. Available at `http://www.w3.org/TR/rdf-concepts/`.

[KLPZ04] M. Kifer, R. Lara, A. Polleres, and C. Zhao. A logical framework for web service discovery. In *Semantic Web Services Workshop*, November 2004.

[KLW95] M. Kifer, G. Lausen, and J. Wu. Logical foundations of object-oriented and frame-based languages. *Journal of ACM*, 42:741–843, July 1995.

[LGF03] N. Li, B.N. Grosof, and J. Feigenbaum. Delegation logic: A logic-based approach to distributed authorization. *ACM Transactions on Information Systems Security*, 6(1), 2003.

[Llo87a] John W. Lloyd. *Foundations of logic programming (second, extended edition)*. Springer series in symbolic computation. Springer-Verlag, New York, 1987.

[Llo87b] John W. Lloyd. *Foundations of logic programming (second, extended edition)*. Springer series in symbolic computation. Springer-Verlag, New York, 1987.

[LMR92] J. Lobo, J. Minker, and A. Rajasekar. *Foundations of Disjunctive Logic Programming*. MIT Press, Cambridge, Massachusetts, 1992.

[LPF+05] Nicola Leone, Gerald Pfeifer, Wolfgang Faber, Thomas Eiter, Georg Gottlob, Simona Perri, and Francesco Scarcello. The DLV System for Knowledge Representation and Reasoning. *ACM Transactions on Computational Logic*, 2005. To appear. Available via `http://www.arxiv.org/ps/cs.AI/0211004`.

[LR98] Alon Y. Levy and Marie-Christine Rousset. Combining Horn rules and description logics in CARIN. *Artificial Intelligence*, 104(1–2):165–209, 1998.

[Mah01] M.J. Maher. Propositional defeasible logic has linear complexity. *TPLP*, 1(6):691–711, 2001.

[Mah02] M.J. Maher. A model-theoretic semantics for defeasible logic. In *Proc. Paraconsistent Computational Logic*, volume 95, pages 67–80. Datalogiske Skrifter, 2002.

[Mar] Massimo Marchiori. Metalog - towards the Semantic Web. Available at `http://www.w3.org/RDF/Metalog/`.

[MG99] M.J. Maher and G. Governatori. A semantic decomposition of defeasible logics. In *Proc. AAAI 99*, pages 299–305, 1999.

[MM04] Frank Manola and Eric Miller. RDF Primer. W3C Recommendation, 10 February 2004. Available at `http://www.w3.org/TR/2004/REC-rdf-primer-20040210/`.

[MRA+01] M.J. Maher, A. Rock, G. Antoniou, D. Billington, and T. Miller. Efficient defeasible reasoning systems. *International Journal of Tools with Artificial Intelligence*, 10(4):483–501, 2001.

[MSS04a] Boris Motik, Ulrike Sattler, and Rudi Studer. Query Answering for OWL-DL with Rules. In *International Semantic Web Conference 2004*, volume 3298 of *LNCS*, pages 549–563. Springer-Verlag, 2004.

[MSS04b] Boris Motik, Ulrike Sattler, and Rudi Studer. Query answering for owl-dl with rules. In Sheila A. McIlraith, Dimitris Plexousakis, and Frank van Harmelen, editors, *International Semantic Web Conference*, volume 3298 of *Lecture Notes in Computer Science*, pages 549–563. Springer, 2004.

[MT93] V.W. Marek and M. Truszczynski. Nonmonotonic logics; context dependent reasoning. In *??* Springer-Verlag, 1993.

[MV03] Boris Motik and Raphael Volz. Optimizing query answering in description logics using disjunctive deductive databases. In François Bry, Carsten Lutz, Ulrike Sattler, and Mareike Schoop, editors, *KRDB*, volume 79 of *CEUR Workshop Proceedings*. Technical University of Aachen (RWTH), 2003.

[NS97] N.Lindenstrauss and Y. Sagiv. Automatic termination analysis of logic programs. In *International Conference on Logic Programming*, 1997.

[Ont] Ontoprise, GmbH. Ontobroker manual. http://www.ontoprise.com/.

[Prz91] T. C. Przymusinski. Stable semantics for disjunctive logic programs. *"New Generation Computing"*, 9:401–424, 1991.

[Prz94] T. Przymusinski. Well-founded and stationary models of logic programs. *Annals of Mathematics and Artificial Intelligence*, 12:141–187, 1994.

[PSHH03] Peter F. Patel-Schneider, Patrick Hayes, and Ian Horrocks. OWL web ontology language semantics and abstract syntax. W3C Candidate Recommendation, 18 August 2003. Available at `http://www.w3.org/TR/owl-semantics/`.

[PSMB+91] Peter F. Patel-Schneider, Deborah L. McGuiness, Ronald J. Brachman, Lori Alperin Resnick, and Alexander Borgida. The CLASSIC knowledge representation system: Guiding principles and implementation rational. *SIGART Bull.*, 2(3):108–113, 1991.

[RDF] The Resource Description Framework. Available at `http://www.w3.org/RDF/`.

[RLK04] D. Roman, H. Lausen, and U. Keller. Web services modeling ontology. Technical report, DERI, November 2004. http://www.wsmo.org/2004/d2/.

[Roo] Jos De Roo. Euler proof mechanism. Available at `http://www.agfa.com/w3c/euler/`.

[Ros99] R. Rosati. Towards expressive KR systems integrating Datalog and Description Logics. In *Proc. 1999 Int. Workshop on Description Logics (DL-1999)*, pages 160–164, 1999.

[RWG02] Daniel M. Reeves, Michael P. Wellman, and Benjamin N. Grosof. Automated negotiation from declarative contract descriptions. *Computational Intelligence*, 18(4), 2002.

[Sch89] Manfred Schmidt-Schauß. Subsumption in KL-ONE is undecidable. In Ron J. Brachman, Hector J. Levesque, and Ray Reiter, editors, *Proc. of the 1st Int. Conf. on the Principles of Knowledge Representation and Reasoning (KR'89)*, pages 421–431. Morgan Kaufmann, Los Altos, 1989.

[Sch91] Klaus Schild. A correspondence theory for terminological logics: Preliminary report. In *Proc. of IJCAI'91*, pages 466–471, 1991.

[SDH02] Michael Sintek, Stefan Decker, and Andreas Harth. TRIPLE - A Query, Inference, and Transformation Language for the Semantic Web. In *International Semantic Web Conference (ISWC)*, Sardinia, June 2002. Available at `http://triple.semanticweb.org/`.

[SM00] S. Staab and A. Maedche. Knowledge portals: Ontologies at work. *The AI Magazine*, 22(2):63–75, 2000.

[SSD94] K. Sagonas, T. Swift, and D. S. Warren. XSB as an efficient deductive database engine. In R. T. Snodgrass and M. Winslett, editors, *Proc. of the 1994 ACM SIGMOD Int. Conf. on Management of Data (SIGMOD'94)*, pages 442–453, 1994.

[Swi04] Terrance Swift. Deduction in Ontologies via ASP. In *Logic Programming and Nonmonotonic Reasoning*, volume 2923 of *LNCS*, pages 275–288. Springer-Verlag, 2004.

[SWM03] Michael K. Smith, Chris Welty, and Deborah L. McGuinness. OWL web ontology language guide. W3C Candidate Recommendation, 18 August 2003. Available at `http://www.w3.org/TR/owl-guide/`.

[The03] The DAML Services Coalition. Daml-s: Semantic markup for web services, May 2003. Available at `http://www.daml.org/services/daml-s/0.9/daml-s.html`.

[Tim00] Extensible Markup Language (XML) 1.0 (Second Edition). W3C Recommendation, 6 October 2000. Available at `http://www.w3.org/TR/REC-xml`.

[TRBH04] Dmitry Tsarkov, Alexandre Riazanov, Sean Bechhofer, and Ian Horrocks. Using Vampire to reason with OWL. In Sheila A. McIlraith, Dimitris Plexousakis, and Frank van Harmelen, editors, *Proc. of the 2004 International Semantic Web Conference (ISWC 2004)*, number 3298 in Lecture Notes in Computer Science, pages 471–485. Springer, 2004.

[Var97] M. Y. Vardi. Why is modal logic so robustly decidable? In N. Immerman and Ph. Kolaitis, editors, *Descriptive Complexity and Finite Models*. American Mathematical Society, 1997.

[vEK76] M. van Emden and R. Kowalski. The semantics of predicate logic as a programming language. *Journal of ACM*, 4(23):733–742, 1976.

[VRS91] A. Van Gelder, K. Ross, and J. S. Schlipf. The well-founded semantics for general logic programs. *Journal of the ACM*, 38(3):620–650, July 1991.

[Wal] Adrian Walker. Internet Business Logic. Available at `https://www.reengineeringllc.com/index.html`.

[Wie] Jan Wielemaker. SWI-Prolog/XPCE Semantic Web Library. Available at `http://www.swi-prolog.org/`.

[YK00] G. Yang and M. Kifer. Implementing an efficient DOOD system using a tabling logic engine. In *First International Conference on Computational Logic, DOOD'2000 Stream*, July 2000.

[YK02] G. Yang and M. Kifer. Well-founded optimism: Inheritance in frame-based knowledge bases. In *Intl. Conference on Ontologies, DataBases, and Applications of Semantics for Large Scale Information Systems (ODBASE)*, October 2002.

[YK03a] G. Yang and M. Kifer. Inheritance and rules in object-oriented semantic web languages. In *Rules and Rule Markup Languages for the Semantic Web (RuleML03)*, volume 2876 of *Lecture Notes in Computer Science*. Springer Verlag, November 2003.

[YK03b] G. Yang and M. Kifer. Reasoning about anonymous resources and meta statements on the semantic web. *Journal on Data Semantics, LNCS 2800*, 1:69–98, September 2003.

[YKZ02] G. Yang, M. Kifer, and C. Zhao. $\mathcal{F}$LORA-2: User's Manual. http://flora.sourceforge.net/, June 2002.

[YKZ03] G. Yang, M. Kifer, and C. Zhao. Flora-2: A rule-based knowledge representation and inference infrastructure for the semantic web. In *International Conference on Ontologies, Databases and Applications of Semantics (ODBASE-2003)*, November 2003.