Reasoning Paradigms for OWL Ontologies

Jing Mei
Department of Information Science
Peking University
Beijing 100871, China
email:mayyam@is.pku.edu.cn

Elena Paslaru Bontas
Freie Universität Berlin
Institut für Informatik
AG Netzbasierte Informationssysteme
Takustr.9, D-14195 Berlin, Germany
email:paslaru@inf.fu-berlin.de

November 29, 2004

Technical Report B-04-12

Abstract

Representing knowledge in OWL provides two important limitations; on one hand efficient reasoning on real-world ontologies containing a large set of individuals is still a challenging task. On the other hand though OWL offers a reasonable trade-off between expressibility and decidability, it can not be used efficiently to model certain application domains. In this paper we give an overview of some of the most relevant approaches in this domain and present OWL2Jess, which is a comprehensive converter tool enabling Jess reasoning over OWL ontologies.

Contents

1 Introduction

The Web Ontology Language OWL[PSHH04] has became a W3C Recommendation in February 2004, and building OWL ontologies is supported by common ontology editors. However representing knowledge in OWL provides two important limitations; on one hand efficient reasoning on real-world ontologies containing a large set of individuals is still a challenging task. On the other hand though OWL offers a reasonable trade-off between expressibility and decidability, it can not be used efficiently to model certain application domains.

Therefore the Semantic Web community tries to overcome these drawback by proposing two directions in dealing with OWL and corresponding reasoning engines: efficient reasoning over individuals could be achieved by identifying fragments of OWL which can be easily translated to F-Logic, while the limited expressibility is extended by Semantic Web-enabled rule languages.

In fact, in classical Artificial Intelligence, both standard ontology languages, which inherit from frame-based systems or semantic networks, and rule languages are similar FOL-related formalisms in Knowledge Representation. Rule-based languages rely on predicate logics, taking advantage of its well-defined semantics and well-understood and powerful inference mechanism[BG94]. Ontology languages like OWL or DAML+OIL rely on F-Logic and Description Logics and can be used for certain modelling tasks in a more intuitive manner. Besides, Description Logics do provide efficient computational properties w.r.t. to automatic classification and consistency checking. The semantic discrepancy of ontology languages and rule-based languages is still an open issue. Each representation paradigm should be used for the particular types of knowledge representation and reasoning tasks it better fits while hybrid reasoning frameworks could be a solution to the heterogeneous representation.

In this report we propose OWL2Jess, a hybrid reasoning framework (Figure-1), which puts these ideas into practice. The tool can be used to fill the gap between OWL and Jess[1], the Java Expert System Shell, a Java-based rule engine and scripting environment. While domain knowledge is still modelled with OWL, using common ontology editors, we transform this OWL formalism to Jess facts using XSL transformations on the XML-syntax of OWL and represent additional rules in Jess. In addition to our predefined entailment rules on the basis of RDF semantics[HM04] and OWL (RDF-compatible) Model-Theoretic Semantics[PSHH04], we run the Jess rule engine to implement the reasoning services.

Note that the predefined rules are used to check the consistency, to compute the taxonomic classification, the characteristics of RDF/OWL vocabulary etc. The inferred Jess assertions are helpful for the ontology engineer to evaluate and refine the original OWL ontology.

2 Preliminary

Before describing the OWL2Jess tool we give an overview of several preliminaries, necessary to explain and motivate our approach. First we will give a short introduction to rule-based languages, including the Rule Markup Language(RuleML) and the Semantic Web Rule Language(SWRL) whose initiative

[1] http://herzberg.ca.sandia.gov/jess/

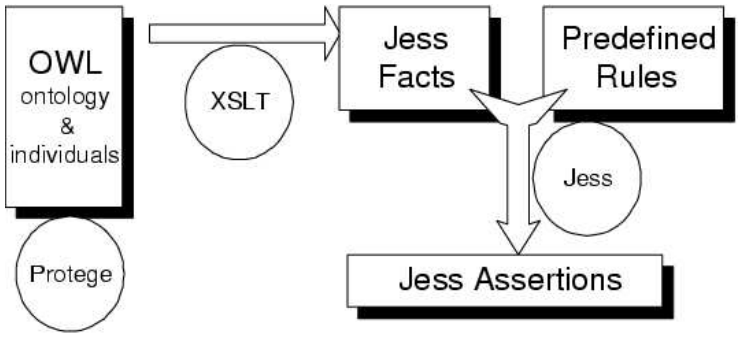

Figure 1: OWL2Jess Model

are offer a Semantic Web compatible representation of rules on XML basis. Further on we present several tools and reasoning engines for common rule languages and stress the possibility of using them on the Semantic Web.

2.1 Classical Rule based Languages

In rule-based languages, terms are built as in the corresponding first-order language, and atoms have the form $p(t_1, \cdots, t_n)$, where the t_i's are terms and p is a predicate symbol of arity n. A literal L has either the form A or $\neg A$ where A is an atom and '$\neg$' is a logical connective called classical negation.

A program is a finite set of rules having the form (*):

$$l_0 \vee \cdots \vee l_k \leftarrow l_{k+1} \wedge \cdots \wedge l_m \wedge \sim l_{m+1} \wedge \cdots \wedge \sim l_n \quad (*)$$

- if l_i's are literals, it is an $AnsProlog^{+(\vee),+(\neg)}$ program.
- if $k = 0, l_i$'s are atoms, it is an $AnsProlog$ program, i.e., a **normal** (or, general) program.
- if $k = 0, m = n, l_i$'s are atoms, it is an $AnsProlog^{-(\sim)}$ program, i.e., a **definite** program, and the rule is known as a Horn rule.

- equality-free rule: the equality predicate does not appear in it.
- safe rule: all variables in the head also occur in the body.
- Datalog: a safe Horn rule with no logical functions.

'$\sim$' is a logical connective called negation-as-failure (naf) or default negation, which should be distinguished from the classical '$\neg$'. By the epistemic interpretation of logic programs, '$\neg p$' can be interpreted as "believe that p is false" while '$\sim p$' as "there is no reason to believe in p".

The semantics of standard rule languages are closely related to Herbrand model semantics. Answer set/stable model semantics is a good choice when

negation-as-failure is taken into account(i.e., n>m). Every answer set/stable model is a minimal Herbrand model, but not all minimal Herbrand models are answer sets/stable models[BG94].

Several rule languages and reasoning frameworks have emerged in the area of logic programming. Prolog is a generalized name which is short for PROgramming in LOGic. AnsProlog(or Answer Set Programming) is a logic programming language based on answer sets/stable model semantics. The inference mechanism of Prolog is based upon Robinson's resolution principle, while AnsProlog takes aim at computing models and answer sets, rather than resolution refutation proofs, i.e., problem solutions are represented by answer sets of the program, but not by variable substitutions[MLYL04]. F-logic[KLW95] (Frame Logic) was proposed to provide a logical foundation for frame-based and object-oriented languages for data and knowledge representation. It has features including object identity, complex objects, inheritance, polymorphic types, query methods, encapsulation etc. It is suitable for defining, querying and manipulating database schema. In a sense, F-logic stands in the same relationship to the object-oriented paradigm as classical predicate calculus stands to relational programming.

2.2 Rule Languages for the Semantic Web

The Rule Markup Initiative[2] is taking steps towards defining a shared Rule Markup Language(RuleML), permitting both forward (bottom-up) and backward (top-down) rules in XML for deduction, rewriting, and further inferential-transformational tasks. Their main objective is to provide a basis for an integrated rule-markup approach that will be beneficial for Semantic Web applications. The RuleML core language can serve as a specification for immediate rule interchange and can be gradually extended.

The Specification of RuleML 0.87, which has been released recently, starts with Datalog to test and refine some principal strategy, and then extends the core language to further declarative rules as allowed in (equational) Horn logic. Corresponding to Semantic Web, a URL/URI module is also introduced, and the 'UR'-Datalog join of both of these classes permits inferences over RDF-like 'resources' and can be re-specialized to RDF triples. The family of RuleML rule languages is illustrated in Figure-2.

Note that the XML Schema negation module, though provided in the specification of RuleML 0.87, is not part of Figure-2. This negation module defines the classical negation 'Neg' and the negation-as-failure 'Naf'. Comparing to the form (*), it firstly makes sure of being safe and equality-free as well as having no functions. Further on by redefining the Datalog module, the XML Schema 'negdatalog' serves as the form (*) where $k = 0, m = n, l_i$'s are literals, 'nafdatalog' as the form (*) where $k = 0, m \leq n, l_i$'s are atoms, and 'nafnegdatalog' as the form (*) where $k = 0, m \leq n, l_i$'s are literals.

Recently, First-Order-Logic RuleML (FOL RuleML) has been released, which is syntactically characterized by explicit quantifiers and head disjunctions, as well as equivalence and negation. On the other side, SWRL[HPSB$^+$04] has been proposed as an extension of the OWL language to include Horn-like rules. SWRL FOL[PS04] is a SWRL extension to First-Order Logic. SWRL FOL and

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

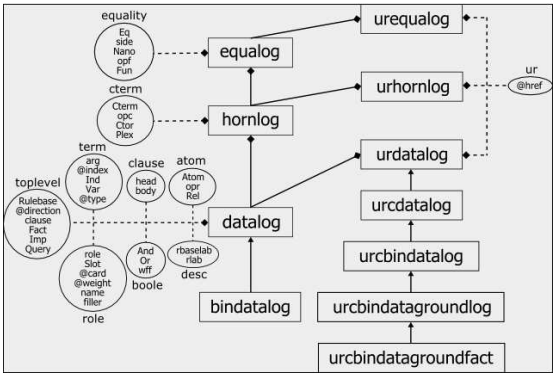

Figure 2: RuleML Modularization

FOL RuleML can be joined to SWRL FOL RuleML by either adding RuleML extensions such as n-ary relations to SWRL FOL or by adding SWRL extensions such as data-valued properties to FOL RuleML.

3 Reasoning Tools for the Semantic Web

There are two groups of currently commercially important rule-based systems. One group primarily employs forward chaining, and their applications heavily rely on their capabilities for procedural attachments. Production rule systems descended from OPS5 as well as event-condition-action (ECA) rule systems are the representatives of this so-called "reactive" category of systems. The other group utilizes backward chaining, i.e., query-answering, and is sometimes called "derivational". This group is mainly comprised of Prolog systems, together with SQL-type relational database systems, whose core are relational algebra and Datalog[GGF02]. Further systems support both forward chaining and backward chaining. In the following, we will give a rough description of some systems without a strict distinction of their types.

3.1 Rule Engines

3.1.1 Jess

Jess[3](Java Expert System Shell),originally inspired by the CLIPS expert system shell and the CLIPS language(C Language Integrated Production System) is a productive development and delivery expert system tool. It provides a complete environment for the construction of rule and/or object based expert systems. Like CLIPS, Jess uses the Rete (Latin for "net") algorithm[For82] to process rules, whose main idea is to improve the speed of forward-chained rule systems by limiting the effort required to recompute the conflict set after a rule is fired. However, Jess also adds many features to CLIPS, including backwards chaining, working memory queries, and the ability to manipulate and directly reason about Java objects. In short, Jess has grown into a complete, distinct, dynamic environment of its own, with powerful Java scripting capabilities.

[3]http://herzberg.ca.sandia.gov/jess/

In Jess, a rule consists of a left-hand side(LHS), the symbol "⇒", and a right-hand side(RHS), in that order. The LHS is made up of zero or more conditional elements, while the RHS consists of zero or more function calls. A conditional element is either a pattern, or a grouping construct like 'and', 'or', or 'not'. The conditional elements are matched against Jess's working memory. When they all match, and if the engine is running, the code on the rule's RHS will be executed. Essentially, a Jess function is a Java method that gets called as a procedure, rather than a true constructor.

Comparing to the above form (*), a Jess program is similar to a general program (i.e., $k = 0, m \leq n, l_i$'s are atoms) where the 'not' conditional element acts as negation-as-failure. However, Jess does not apply any answer set/stalbe model semantics. Furthermore, although more than one elements are permitted in the RHS of a Jess rule, it just means $l_0 \wedge \cdots \wedge l_k$ but not $l_0 \vee \cdots \vee l_k$, since all rather than one of these function calls would be executed.

3.1.2 XSB

XSB[4] is a Logic Programming and Deductive Database system, and it extends Prolog with new semantic and operational features, mostly based on the use of Tabled Logic Programming or tabling. Consequently, it was also named after Tabled Prolog. XSB can be used as a Prolog system since it provides all the functionality of Prolog. Further on XSB is based on tableaux calculus, which provides termination to various classes of programs including definite programs(also called Horn Clause Programs) and normal programs. Besides, both backtrackable and non-backtrackable updates to asserted code are included in XSB.

3.1.3 Cwm

Cwm[5] is a general-purpose data processor for the Semantic Web, which is part of SWAP (Semantic Web Application Platform). Cwm is a forward chaining reasoner which can be used for querying, checking, transforming and filtering information. Its core language is RDF, extended to include rules. It uses RDF/XML or RDF/N3 serializations as required. We note that Cwm was designed as a proof of concept for the standard Semantic Web layered architecture. Therefore the implementation of the reasoner does not particularly focus on scalability or performance issues for large data sets or rule sets as in real-world applications.

3.1.4 RuleML Engines

Mandarax[6] was implemented as the first complete input-processing-output environment for RuleML. In Mandarax, rule bases can be made persistent using the XKB module which stored rules and other knowledge in a format similar to RuleML, and export and import of RuleML rule bases are also supported. Being pure OO, Mandarax is an open source java class library for deduction rules, and it is based on backward reasoning. jDREW[7] also provides modules to process rules in RuleML format, and it is an deductive reasoning engine for

[4] http://xsb.sourceforge.net/
[5] http://www.w3.org/2000/10/swap/doc/cwm.html
[6] http://mandarax.sourceforge.net/
[7] http://www.jdrew.org/jDREWebsite/jDREW.html

clausal first order logic (facts and rules) written in Java and well integrated with the Web. Besides, XSLT translators can directly transform the RuleML rulebases into the representation for a specified target engine such as XSB, Jess and Cwm.

3.2 Rules-Enabled Ontology Engineering Platforms

After describing classical rule engines we concentrate on ontology engineering platforms emerged in the setting of the Semantic Web, which support to some extent rule-based modelling and reasoning services. Table-1 summarizes the main features of the four systems we analyze in this section.

Table 1: Comparison

System	Feature	Reasoning
KAON	ontology management	Datalog engine for DLP
FLORA-2	F-Logic	XSB as rule engine
iS	DL reasoner + DB query	TBox in DL, ABox in DB

3.2.1 KAON

KAON[8] is an open-source ontology management infrastructure including a comprehensive tool suite to ease the creation and management of ontologies and to assist the development of ontology-based applications. The main goal of KAON is to retain scalability in reasoning with large ontologies and knowledge bases. Therefore it currently relies on (deductive) database techniques to support efficient reasoning with instances.

DLP (Description Logic Programs) are part of the KAON framework, and can be used to transform OWL ontologies into a (disjunctive) logic program, which is processed using the KAON Datalog engine. DLP-compatible ontologies do not cover the whole range of the OWL language.

3.2.2 FLORA-2

FLORA-2[9] is a sophisticated object-oriented knowledge base language and application development environment, which translates a unified language of F-logic, HiLog, and Transaction Logic into the XSB deductive engine. The tabled resolution is useful for recursive query computation, allowing programs to terminate correctly in many cases where Prolog does not.

3.2.3 Instance Store

The Instance Store(IS)[10] is a Java application for performing efficient and scalable Description Logic reasoning over individuals. The IS system consists of a database to store all assertions over individuals, a reasoner to perform DL reasoning and an ontology to covering TBox knowledge. It defines two basic operations, namely addAssertion(individual, description) and retrieve(description).

[8] http://kaon.semanticweb.org/
[9] http://flora.sourceforge.net/
[10] http://instancestore.man.ac.uk/

In a word, it supports reasoning by means of (relational) databases, reducing the amount of reasoning to pure terminological reasoning.

3.3 OWL Reasoners

A last group of engines is related directly to the ontology languages for the Semantic Web like OWL and RDF(S). A summary of their capabilities is presented in Table-2. Besides, the well-known DL reasoners FaCT and RACER are introduced considering OWL is closely related to DL.

Table 2: OWL Reasoner

Reasoner	Mechanism	Related to
F-OWL	backchaining with tabling in XSB	Flora-2
Euler	resolution with Euler path detection	N3/Cwm
Pellet	OWL DL reasoner	Mindswap
Hoolet	first order prover	WonderWeb
Cerebra	commercial implementation of DL	NI company
ConsVISor	rule-based system for checking	VIS Inc.
RACER	DL reasoner for TBox, ABox, Concrete Domain	RICE
FaCT	DL reasoner for $\mathcal{SHF}$ and $\mathcal{SHIQ}$	Lisp

F-OWL[11] is an ontology inference engine for OWL based on Flora-2 and XSB.

Euler[12] is an inference engine supporting logic based proofs. It is a backward-chaining reasoner and will tell you whether a given set of facts and rules supports a given conclusion. The proof engine uses the resolution inference mechanism and only follows Euler paths (the concept Euler found several hundred years ago) so that endless deductions are avoided. That means that no special attention has to be paid to recursions or to graph merging.

Pellet[13] is an OWL DL reasoner based on the tableaux algorithms developed for expressive Description Logics. After parsing OWL documents into a triple stores, the OWL abstract syntax are separated into TBox(axioms about classes), ABox(assertions about individuals) and RBox(axioms about properties), which are passed to the tableaux based reasoner.

Hoolet[14] is an OWL DL reasoner that uses a First Order Prover to reason about OWL ontologies. Hoolet is implemented using the WonderWeb OWL API for parsing and processing OWL, and the Vampire theorem prover for First Order classical logic. Other reasoners could also be used since the communication with the reasoner is via the TPTP (Thousands of Problems for Theorem Provers) format which is understood by various theorem provers.

Cerebra[15] is a product of Network Inference, and its technology provides a commercial grade, robust, scalable implementation of the DL algorithms that use OWL documents in their native form. These algorithms are encapsulated into a run-time engine that is provided as a service to other applications or services and can respond to queries about ontologies from those applications.

[11] http://fowl.sourceforge.net
[12] http://www.agfa.com/w3c/euler/
[13] http://www.mindswap.org/2003/pellet/
[14] http://owl.man.ac.uk/hoolet/
[15] http://www.networkinference.com/

Furthermore, the Cerebra Server provides links and techniques to reason about data and information not held within a given ontology, by accessing relational data structures or meta-data.

ConsVISor[16] is a Consistency Checker for OWL Full, DL and Lite that emphasizes the generation of highly descriptive error and warning messages in both HTML and OWL formats. ConsVISor is a rule-based system for checking the consistency of ontologies and annotations serialized in RDF or OWL, as well as warning the ontologist about elements that, while not necessarily inconsistent, do not correspond to common modelling practices in OWL or RDF(S).

RACER[17] (Renamed ABox and Concept Expression Reasoner) is a Description Logic reasoning system with support for TBoxes with generalized concept inclusions, for ABoxes and for Concrete domains (e.g., linear (in-)equalities over the reals). To support Semantic Web, RACER is also being used as as a Semantic Web inference engine for developing ontologies, for query answering over RDF documents and wrt. specified RDFS/DAML ontologies, and for registering permanent queries (e.g., for building a document management system) with notification of new results if available (publish-subscribe facility). Recently, RACER has been plugged in Protege for developing ontologies in OWL format, and RICE (RACER Interactive Client Environment) provides the GUI for interactively answering queries and visualizing TBoxes and ABoxes.

FaCT[18] (Fast Classification of Terminologies) is a Description Logic classifier that can also be used for modal logic satisfiability testing. The FaCT system includes two reasoners, one for the logic $\mathcal{SHF}$ ($\mathcal{ALC}$ augmented with transitive roles, functional roles and a role hierarchy) and the other for the logic $\mathcal{SHIQ}$ ($\mathcal{SHF}$ augmented with inverse roles and qualified number restrictions), both of which use sound and complete tableaux algorithms. Noting FaCT is written in Common Lisp, and it would be run on any system where a suitable Lisp is available, however it also provides a CORBA interface to run in a FaCT server and a DIG servlet to run for Windows or Linux platform.

4 OWL2Jess

This chapter introduces a tool which converts OWL ontologies to Jess knowledge bases. According different expressivity levels, it actually could be either reduced to pure RDF2Jess, or extended to newly SWRL2Jess where SWRL[HPSB$^+$04] extends the set of OWL axioms to include Horn-like rules. By converting OWL syntactically and semantically to Jess the tool enables the usage of the Jess reasoning engine over OWL ontologies, which might be more efficient than DL reasoners for particular tasks. Besides, when extending an OWL-formalized knowledge base with rules, one needs a common reasoning engine for the corresponding heterogeneous data base, while preserving the advantages of using OWL for particular modelling tasks.

4.1 Transforming OWL to Jess

Converting OWL knowledge bases to Jess in realized in four steps.

[16] http://www.vistology.com/consvisor
[17] http://www.sts.tu-harburg.de/ r.f.moeller/racer/
[18] http://www.cs.man.ac.uk/ horrocks/FaCT/

1). The first step is to build the ontology. An ontology editor like Protege[19] provides an OWL Plug-in to support the development of OWL ontologies. Organizing knowledge in terms of classes, properties, restrictions and individuals has been proven to be well accepted by domain experts and software developers, since this paradigm is very similar to object-oriented modelling or UML. Besides in the last decades various ontologies for almost every application domain have been formalized in RDF(S) and OWL and can be therefore re-used to be extended with rules if necessary.

2). The second step is to transform the XML syntax of OWL into the Jess syntax by means of XSLT. Starting from the root, recurrent processes of ABox-class and ABox-property are called via a set of named templates. The output file consists of Jess facts. If the semantics of the underlying ontology languages is already specified as Jess rules, specific keyword matching is unnecessary in the OWL2Jess XSL transformation.

```
  <xsl:template name="ABox-property" >
(assert (triple
 (predicate "<xsl:value-of select="concat(ns-uri(.),name(.))"/>")
 (subject   "<xsl:call-template name="get-father-ID"/>")
 (object    "<xsl:call-template name="get-child-ID"/>"))
)<xsl:for-each select="*[position()=1]">
  <xsl:call-template name="ABox-class"/>
 </xsl:for-each></xsl:template>
```

3). The third step is to combine the Jess files, including the result of XSLT, and our predefined RDF/OWL entailment rules. Furthermore, external Jess-style queries and rulers can also be appended, such as the composition of properties (like "hasUncle(x,z) ← hasFather(x,y), hasBrother(y,z)"). Such rules could also be represented using the SWRL rule language and the SWRL2Jess transformation tool.

4). The fourth step is to run the Jess rule engine. Among our predefined rules, we mention consistency checking, classification and characteristics. Output results with error messages indicate invalid or illegal issues in the incoming OWL ontology. Caution messages are used to signal whether the engine has discovered an individual belonging to a certain class, moreover the machine would randomly deal with the uncertainty like $\exists$ or $\vee$, based on the currently given knowledge base.

4.2 Variants of OWL2Jess

Considering OWL (RDF-compatible) Model-Theoretic Semantics[PSHH04] is based on RDF Semantics[HM04], our Jess rule file "owlmt.clp" includes a separate "rdfmt.clp", resulting that a pure RDF document would also be applied using a simpler RDF2Jess model, which will avoid unnecessary, expensive computations on the more complex OWL semantics.

Another possible alternative to OWL2Jess is its extension to support SWRL-enhanced OWL ontologies. Consequently, by another "SWRL2Jess.xsl" stylesheet, the translation from the SWRL syntax to Jess syntax could be easily accomplished. However, unlike the above OWL2Jess XSL transformation, we need

[19] http://protege.stanford.edu/

additional keyword matching template to distinguish among different types of SWRL rules and their components (see the XSLT fragment below). The output consists of Jess rules, which will be handled in the same way as other entailment rules sharing the common ontology knowledge.

```
 <xsl:template match="ruleml:Imp">
(defrule imp <xsl:apply-templates select="ruleml:body" />
         =>  <xsl:apply-templates select="ruleml:head" />
)</xsl:template>

 <xsl:template match="swrl:ClassAtom">
(triple
 (predicate "&rdf;#type")
 (object    <xsl:apply-templates select="swrl:argument1" />)
 (subject   <xsl:apply-templates select="swrl:classPredicate" />)
)</xsl:template>
```

4.3 Some Issues about the OWL to Jess Transformation

As mentioned above, our proposal suggests to build ontologies in OWL and transform the OWL knowledge base to Jess for particular reasoning purposes. However, the semantics of OWL currently adopts an open world assumption(OWA), while all rule-based languages including Jess are based on a close world assumption(CWA).

In OWA, everything which was not specified explicitly is unknown to the reasoning service. For example, an owl:Class is defined as $C = \forall P.D$, but we can not conclude $u \in C$ even if we have currently found out "for any given $v, P(u, v)$ there is $v \in D$". The reason is there are more unknown t, maybe $P(u, t)$ but $t \notin D$. This fact can not be derived automatically due to the open world assumption. However, in practice, we indeed need such real-time conclusions, especially when we want to know whether there is something wrong or something missing about our existing ontology.

Consequently, our intention is merely to check the given OWL ontology, to remind the errors or cautions, and to suggest the modifications, rather than attempt to modify it. According our error or caution messages, the author can revise the ontology manually. Furthermore, the inferred Jess assertions are helpful for the author to recognize all characteristics of the ontology, such as that an individual currently belongs to an OWL restriction or an OWL boolean combination, even if this assertion is missing from the original ontology.

Another important issue is related to "if-and-only-if" (iff) conditions. In RDF Semantics[HM04], there are extensional semantic conditions (i.e., iff conditions) for rdfs:subClassOf, rdfs:subPropertyOf, rdfs:domain, and rdfs:range, which are so strong that some consequences inferred are useless in practice. For instance, the domain and range of every property are extended to the largest one, namely rdfs:Resource, resulting in confusions with original definitions. A similar situation appears for the OWL (RDF compatible) Semantics[PSHH04]. We ignore these extensions here, however they are easy to be included if they are required.

5 Entailment Rules

According to the RDF Semantics[HM04] and the OWL (RDF-compatible) Model-Theoretic Semantics[PSHH04], we implement the entailment rules in Jess one by one. In the following we present the most important ones with a focus on owl:Restrictions and boolean expressions, which are viewed as expressive restrictions [GHVD03] for rule-based languages.

Some works-around are helpful to cope with the semantic discrepancy between OWL and rule-based languages: error messages indicating some illegal or invalid issues in the ontology or caution messages pointing out potentially missing ontology statements.

5.1 RDF Semantics

RDF(S) axiomatic triples are transformed to facts. The following is a simple Jess fact for "rdf:type rdf:type rdf:Property".

```
(deffacts RDF_axiomatic_triples
  (triple (predicate "rdf:type")
          (subject   "rdf:type")
          (object    "rdf:Property")))
```

It is unnecessary to assert the triples mentioned as RDFS-valid, such as "rdfs:Class rdf:type rdfs:Class", since all these could be inferred by other existing rules.

RDF(S) semantic conditions are transformed to rules, some of which are presented as follows. We ignore the extensional semantic conditions, i.e., if-and-only-if conditions for rdfs:subClassOf, rdfs:subPropertyOf, rdfs:domain, and rdfs:range, which are so strong that some consequences inferred are useless in practice. For instance, the domain and range of every property are extended to the largest one, namely rdfs:Resource, which would confuse with our intention. Of course, the same situations also appear in OWL semantics, however they are easy to be included if they are required.

5.1.1 rdf:type

It is stated as a basic RDFS semantic condition that, x is in ICEXT(y) if and only if $< x, y >$ is in IEXT(I(rdf:type)). Thus, we translate the inclusion relation into a rule as below:

```
(defrule RDFS_semantic_conditions_type
  (triple (predicate "rdf:type") (subject ?s) (object ?o))
  =>
  (assert (triple (predicate "rdf:type")
                  (subject   ?o)
                  (object    "rdfs:Class"))))
```

5.1.2 rdfs:domain

It is stated in RDF Semantics that, if $< x, y >$ is in IEXT(I(rdfs:domain)) and $< u, v >$ is in IEXT(x) then u is in ICEXT(y). The corresponding rule is as follows:

```
(defrule RDFS_semantic_conditions_range
  (triple (predicate "rdfs:domain") (subject ?x) (object ?y))
  (triple (predicate ?x) (subject ?u) (object ?v))
  =>
  (assert(triple (predicate "rdf:type") (subject ?u) (object ?y))))
```

5.1.3 rdfs:range

It is stated in RDF Semantics that, if $< x, y >$ is in IEXT(I(rdfs:range)) and $< u, v >$ is in IEXT(x) then v is in ICEXT(y). The corresponding rule is as follows:

```
(defrule RDFS_semantic_conditions_range
  (triple (predicate "rdfs:range") (subject ?x) (object ?y))
  (triple (predicate ?x) (subject ?u) (object ?v))
  =>
  (assert(triple (predicate "rdf:type") (subject ?v) (object ?y))))
```

5.1.4 rdfs:subPropertyOf

First rdfs:subPropertyOf is transitive and reflexive.

```
(defrule RDFS_semantic_conditions_subPropertyOf_transitive
  (triple (predicate "rdfs:subPropertyOf") (subject ?x) (object ?y))
  (triple (predicate "rdfs:subPropertyOf") (subject ?y) (object ?z))
  =>
  (assert (triple (predicate "rdfs:subPropertyOf")
                  (subject ?x) (object ?z))))

(defrule RDFS_semantic_conditions_subPropertyOf_reflexive
  (triple
   (predicate "rdf:type")
   (subject   ?p)
   (object    "rdf:Property"))
   =>
  (assert (triple (predicate "rdfs:subPropertyOf")
                  (subject ?p) (object ?p))))
```

Moreover it has the following characteristics:

```
(defrule RDFS_semantic_conditions_subPropertyOf
   (triple (predicate "rdfs:subPropertyOf") (subject ?x) (object ?y))
   (triple (predicate ?x) (subject ?a) (object ?b))
   =>
   (assert (triple (predicate ?y) (subject ?a) (object ?b))))
```

5.1.5 rdfs:subClassOf

rdfs:subClassOf has the same properties as rdfs:subPropertyOf.

```
(defrule RDFS_semantic_conditions_subClassOf_transitive
  (triple (predicate "rdfs:subClassOf") (subject ?x) (object ?y))
```

```
  (triple (predicate "rdfs:subClassOf") (subject ?y) (object ?z))
  =>
  (assert (triple (predicate "rdfs:subClassOf")
                  (subject ?x) (object ?z))))

(defrule RDFS_semantic_conditions_subClassOf_reflexive
  (triple
   (predicate "rdf:type")
   (subject   ?c)
   (object    "rdfs:Class"))
   =>
  (assert (triple (predicate "rdfs:subClassOf")
                  (subject ?c) (object ?c))))

(defrule RDFS_semantic_conditions_subClassOf
   (triple (predicate "rdfs:subClassOf") (subject ?x) (object ?y))
   (triple (predicate "rdf:type") (subject ?o) (object ?x))
   =>
   (assert (triple (predicate "rdf:type") (subject ?o) (object ?y))))
```

Moreover, it was pointed out that, any class is a subclass of the largest class, namely rdfs:Resource.

```
(defrule RDFS_semantic_conditions_subClassOf_Resource
  (triple (predicate "rdf:type")
          (subject    ?x)
          (object    "rdfs:Class"))
  =>
  (assert (triple (predicate "rdfs:subClassOf")
                  (subject   ?x)
                  (object    "rdfs:Resource"))))
```

5.2 OWL Semantics

The implementation of the OWL semantics are more challenging, since OWL is an extension of RDFS to provide restrictions on how properties behave in a local class scope[HPSvH03]. We define additional facts to represent typical OWL primitives and their relationship to RDFS such as owl:Class is subclass of rdfs:Class. A set of rules are defined to represent characteristics of OWL classes, datatypes, properties and restrictions. In the remaining of this section we restrict to OWL restrictions and boolean expressions, which are in our opinion the most relevant for the OWL to Jess conversion.

5.2.1 OR condition

We firstly encounter the "or" condition in the head of a rule, since subject-values for owl:AnnotationProperty are owl:Thing or rdfs:Literal. Subsequently $B \vee C \leftarrow A$ equals to $\neg A \vee B \vee C$ equals to $C \leftarrow A \wedge \neg B$, while the first one cannot be expressed in Jess and the last one can. However, when we represent it as follows, by default the subject-values are owl:Thing if there is no definition

in advance. Usually such precise definitions are provided by the ontology author in advance or might be suggested by ontology editors.

```
(defrule OWL_characteristics_AnnotationProperty_object
  (triple (predicate "rdf:type")
          (subject   ?e)
          (object    "owl:AnnotationProperty"))
  (triple (predicate ?e) (subject ?x) (object ?y))
  (not (triple (predicate "rdf:type")
               (subject   ?y)
               (object    "rdfs:Literal")))
  =>
  (assert (triple (predicate "rdf:type")
                  (subject   ?y)
                  (object    "owl:Thing")))
  (printout t "Caution!" ?y " now is in owl:Thing" crlf))
```

5.2.2 EQUIVALENT condition

owl:equivalentClasses and owl:equivalentProperties are stated to be subclasses or subproperties of each other.

```
(defrule OWL_characteristics_equivalentProperty_relationship
  (triple (predicate "owl:equivalentProperty") (subject ?x) (object ?y))
  (test (neq 0 (str-compare ?x ?y)))
  =>
  (assert (triple (predicate "owl:subPropertyOf")
                  (subject ?x) (object ?y)))
  (assert (triple (predicate "owl:subPropertyOf")
                  (subject ?y) (object ?x))))
```

In order to deal with the owl:disjointWith constraint, we make use of owl:differentFrom to state the individuals of the two classes are different from each other.

```
(defrule OWL_characteristics_disjointWith_relationship
  (triple (predicate "owl:disjointWith") (subject ?x) (object ?y))
  (triple (predicate "rdf:type") (subject ?o1) (object ?x))
  (triple (predicate "rdf:type") (subject ?o2) (object ?y))
  =>
  (assert (triple (predicate "owl:differentFrom")
                  (subject ?o1) (object ?o2))))
```

The owl:differentFrom and owl:sameAs constructs can not be translated directly to Jess. Checking $x \neq y$ or $x = y$ is easy, but no further statement could be provided about its meaning. Consequently, an error message is generated to signal this issue to the user.

```
(defrule OWL_characteristics_differentFrom
  (triple (predicate "owl:differrentFrom") (subject ?x) (object ?y))
  (test (eq 0 (str-compare ?x ?y)))
  =>
```

```
  (printout t "Error!" ?x " is not different from " ?y crlf))

(defrule OWL_characteristics_sameAs
  (triple (predicate "owl:sameAs") (subject ?x) (object ?y))
  (test (neq 0 (str-compare ?x ?y)))
  =>
  (printout t "Error!" ?x " is not same as " ?y crlf))
```

5.2.3 owl:complementOf

owl:complementOf is a subproperty of owl:disjointWith. It should be noticed that owl:complementOf has a very strong semantics, resulting in any individual must be in a class or in its complement, once the individual has been stated in the universe. We state the meaning of owl:complementOf by this rule:

```
(defrule OWL_complementOf
  (triple (predicate "owl:complementOf") (subject ?x) (object ?y))
  (triple (predicate "rdf:type") (subject ?u) (object "owl:Thing"))
  (not (triple (predicate "rdf:type") (subject ?u) (object ?y)))
  =>
  (assert (triple (predicate "rdf:type") (subject ?u) (object ?x)))
  (printout t "Caution!" ?u " now is in " ?x crlf))
```

5.2.4 owl:intersectionOf

In OWL, the subject-value of owl:intersectionOf is a sequence of rdf:first, rdf:rest constructs. However, via XSLT, we directly catch a owl:Class as the subject-value, hence we skip the verbose syntax of rdf:List. Moreover, in Set Theory, set equation $A = B$ means $A \subseteq B$ and $A \supseteq B$, so we decompose the set equation of owl:intersectionOf into two rules "subset" and "supset". Surely owl:unionOf and owl:oneOf can be treated in a similar way.

Suppose $\langle x, y\rangle \in EXT_I(S_I(\text{owl:intersectionOf}))$ and y is a sequence of $y_1, \cdots, y_n$, the subset relation is easy to state, because $CEXT_I(x) \subseteq CEXT_I(y_1) \cap \cdots \cap CEXT_I(y_n) \subseteq CEXT_I(y_i)$. That is, for any $u \in CEXT_I(x)$, we have $u \in CEXT_I(y_i), 1 \leq i \leq n$, as shown below.

```
(defrule OWL_intersectionOf_subset
  (triple (predicate "owl:intersectionOf") (subject ?x) (object ?y))
  (triple (predicate "rdf:type") (subject ?u) (object ?x))
  =>
  (assert (triple (predicate "rdf:type") (subject ?u) (object ?y))))
```

Furthermore, the supset relation is also permitted in Jess. The troublesome issue is to check out the individual u who belongs to all subclasses, i.e., $\forall y_i, u \in CEXT_I(y_i)$. Here, we make use of an implication in the body of a rule, namely $C \leftarrow (B \leftarrow A)$, equals to $C \leftarrow (\neg A \vee B)$. However, it is better to write as $\neg(A \wedge \neg B)$ in Jess, for the negation of Jess is the failure of matching, and what we want to know is the result of the matching of B rather A.

```
(defrule OWL_intersectionOf_supset
  (triple (predicate "owl:intersectionOf") (subject ?x) (object ?y))
```

```
  (triple (predicate "rdf:type") (subject ?u) (object ?y))
  (not (and (triple (predicate "owl:intersectionOf")(subject ?x)(object ?v))
            (not (triple (predicate "rdf:type")(subject ?u)(object ?v)))))
  =>
  (assert (triple (predicate "rdf:type") (subject ?u) (object ?x))))
```

5.3 owl:unionOf

If $\langle x, y\rangle \in EXT_I(S_I(\text{owl:unionOf}))$ and y is a sequence of $y_1, \cdots, y_n$, we can easily state the supset relation, because $CEXT_I(x) \supseteq CEXT_I(y_1) \cup \cdots \cup CEXT_I(y_n) \supseteq CEXT_I(y_i)$.

```
(defrule OWL_unionOf_supset
  (triple (predicate "owl:unionOf") (subject ?x) (object ?y))
  (triple (predicate "rdf:type") (subject ?u) (object ?y))
  =>
  (assert (triple (predicate "rdfs:type") (subject ?u) (object ?x))))
```

However, the subset relation is troublesome due to the uncertainty. Given an individual $u \in CEXT_I(x)$, if we check out $\forall y_i, u \notin CEXT_I(y_i)$, then it indicates something missing about the subclasses of the union x in the existing ontology, else $u \in CEXT_I(y_i)$ has been satisfiable. The engine would randomly assign u to one y_i after generating a caution message, which suggests the ontology author to assert a new individual belonging to x and certain y_i in the original ontology.

```
(defrule OWL_unionOf_subset
  (triple (predicate "owl:unionOf") (subject ?x) (object ?y))
  (triple (predicate "rdf:type") (subject ?u) (object ?x))
  (not (and (triple (predicate "owl:unionOf")(subject ?x)(object ?v))
            (triple (predicate "rdf:type")(subject ?u)(object ?v))))
  =>
  (printout t "Caution!" ?u " now is in " ?y crlf)
  (assert (triple (predicate "rdf:type") (subject ?u) (object ?y))))
```

5.4 owl:oneOf

To some extend, owl:oneOf is similar to owl:unionOf, for $\{y_1, \cdots, y_n\} = \{y_1\} \cup \cdots \cup \{y_n\}$. Consequently in the subset relation, rdf:type is changed into owl:sameAs.

```
(defrule OWL_oneOf_subset
  (triple (predicate "owl:oneOf") (subject ?x) (object ?y))
  (triple (predicate "rdf:type") (subject ?u) (object ?x))
  (not(and(triple (predicate "owl:oneOf")(subject ?x)(object ?v))
        (or(test(eq 0 (str-compare ?u ?v)))
           (triple(predicate "owl:sameAs")(subject ?u)(object ?v)))))
  =>
  (printout t "Caution!" ?u " now is same as " ?y crlf)
  (assert (triple (predicate "owl:sameAs") (subject ?u) (object ?y))))
```

The supset relation is also more simple than in the previous cases:

```
(defrule OWL_oneOf_supset
  (triple (predicate "owl:oneOf") (subject ?x) (object ?y))
  =>
  (assert (triple (predicate "rdf:type") (subject ?y) (object ?x))))
```

5.4.1 owl:allValuesFrom

Suppose $\langle x, y\rangle \in EXT_I(S_I(\text{owl:allValuesFrom}))$ and $\langle x, p\rangle \in EXT_I(S_I(\text{owl:onProperty}))$, the subset relation is $CEXT_I(x) \subseteq \{u|\langle u, v\rangle \in EXT_I(p)$ implies $v \in CEXT_I(y)\}$, which can be easily translated into a Jess rule as below:

```
(defrule OWL_allValuesFrom_subset
  (triple (predicate "owl:allValuesFrom") (subject ?x) (object ?y))
  (triple (predicate "owl:onProperty") (subject ?x) (object ?p))
  (triple (predicate "rdf:type") (subject ?u) (object ?x))
  (triple (predicate ?p) (subject ?u) (object ?v))
  =>
  (assert (triple (predicate "rdf:type") (subject ?v) (object ?y))))
```

The supset relation contains an implication in the body of a rule, in which $B \leftarrow A$ is transformed as $\neg(A \wedge \neg B)$, where $A = \langle u, v\rangle \in EXT_I(p), B = v \in CEXT_I(y)$.

```
(defrule OWL_allValuesFrom_supset
  (triple (predicate "owl:allValuesFrom") (subject ?x) (object ?y))
  (triple (predicate "owl:onProperty") (subject ?x) (object ?p))
  (triple (predicate ?p) (subject ?u) (object ?v))
  (not (and (triple (predicate ?p) (subject ?u) (object ?o))
            (not (triple (predicate "rdf:type") (subject ?o) (object ?y)))))
  =>
  (assert (triple (predicate "rdf:type") (subject ?u) (object ?x))))
```

5.4.2 owl:someValuesFrom

Suppose $\langle x, y\rangle \in EXT_I(S_I(\text{owl:someValuesFrom}))$ and $\langle x, p\rangle \in EXT_I(S_I(\text{owl:onProperty}))$, the supset relation is $CEXT_I(x) \supseteq \{u|\exists\langle u, v\rangle \in EXT_I(p)$ such that $v \in CEXT_I(y)\}$. Once we find out one existence, we can assert it as below.

```
(defrule OWL_someValuesFrom_supset
  (triple (predicate "owl:someValuesFrom") (subject ?x) (object ?y))
  (triple (predicate "owl:onProperty") (subject ?x) (object ?p))
  (triple (predicate ?p) (subject ?u) (object ?v))
  (triple (predicate "rdf:type") (subject ?v) (object ?y))
  =>
  (assert (triple (predicate "rdf:type") (subject ?u) (object ?x))))
```

In order to specify the subset relation, for $\langle u, v\rangle \in EXT_I(p)$, we first find out all possible types s of the individual v, and then check whether y is one possibility of s. If it fails, what we could do is to randomly assign one to belong to y, else $v \in CEXT_I(y)$ has been satisfiable.

```
(defrule OWL_someValuesFrom_subset
  (triple (predicate "owl:someValuesFrom") (subject ?x) (object ?y))
  (triple (predicate "owl:onProperty") (subject ?x) (object ?p))
  (triple (predicate "rdf:type") (subject ?u) (object ?x))
  (triple (predicate ?p) (subject ?u) (object ?v))
  (not (and (triple (predicate ?p) (subject ?u) (object ?o))
            (triple (predicate "rdf:type") (subject ?o) (object ?s))
            (test (eq 0 (str-compare ?s ?y)))))
  =>
  (printout t "Caution!" ?v " now is in " ?y crlf)
  (assert (triple (predicate "rdf:type") (subject ?v) (object ?y))))
```

5.4.3 owl:hasValue

Suppose $\langle x, y\rangle \in EXT_I(S_I(\text{owl:hasValue}))$ and $\langle x, p\rangle \in EXT_I(S_I(\text{owl:onProperty}))$, the set equation is $CEXT_I(x) = \{u | \exists \langle u, v\rangle \in EXT_I(p)\}$, and the translations are also easy.

```
(defrule OWL_hasValue_subset
  (triple (predicate "owl:hasValue") (subject ?x) (object ?y))
  (triple (predicate "owl:onProperty") (subject ?x) (object ?p))
  (triple (predicate "rdf:type") (subject ?u) (object ?x))
  =>
  (assert (triple (predicate ?p) (subject ?u) (object ?y))))
(defrule OWL_hasValue_supset
  (triple (predicate "owl:hasValue") (subject ?x) (object ?y))
  (triple (predicate "owl:onProperty") (subject ?x) (object ?p))
  (triple (predicate ?p) (subject ?u) (object ?y))
  =>
  (assert (triple (predicate "rdf:type") (subject ?u) (object ?x))))
```

5.4.4 owl:cardinality

Suppose $\langle x, y\rangle \in EXT_I(S_I(\text{owl:cardinality}))$ and $\langle x, p\rangle \in EXT_I(S_I(\text{owl:onProperty}))$, the subset relation is $CEXT_I(x) \subseteq \{u | card\{\langle u, v\rangle \in EXT_I(p)\} = y\}$. We can compute the number of v using the Jess function "count-query-results". Error messages are thrown in case the result does not equal to y.

```
(defquery OWL_cardinality_query
    (declare (variables ?P ?S))
    (triple (predicate ?P) (subject ?S) (object ?O) ))
(defrule OWL_cardinality_subset
  (triple (predicate "owl:cardinality") (subject ?x) (object ?y))
  (triple (predicate "owl:onProperty") (subject ?x) (object ?p))
  (triple (predicate "rdf:type") (subject ?u) (object ?x))
  (test (<> ?y (count-query-results OWL_cardinality_query ?p ?u)))
  =>
 (printout t "Error!" ?x " has no " ?y " relating to " ?p crlf))
```

The supset relation is asserted by means of the "count-query-results" function.

```
(defrule OWL_cardinality_supset
  (triple (predicate "owl:cardinality") (subject ?x) (object ?y))
  (triple (predicate "owl:onProperty") (subject ?x) (object ?p))
  (triple (predicate ?p) (subject ?u) (object ?v))
  =>
  (if (= ?y (count-query-results OWL_cardinality_query ?p ?u)) then
   (assert (triple (predicate "rdf:type") (subject ?u) (object ?x)))))
```

6 Evaluation and Future Work

6.1 Test Cases

In a first step all entailment rules are tested. In the output file of "rdfmt.clp", there are 55 predefined assertions, which completely consists of RDF(S) axiomatic triples, and 108 inferred assertions, including the triples mentioned as rdfs-valid, such as (pred sub obj)=(rdf:type rdfs:Class rdfs:Class). In order to support the OWL semantics, this file is extended to 300 assertions, most of which are simple ones resulting from the reflexive rdfs:subClassOf and rdfs:subPropertyOf.

In a second step a small OWL ontology representing the family terminology and some individuals asserting their relationships is tested. Besides the obvious conclusions like those related to owl:Thing, more attention is paid to owl:Restriction and boolean expressions. A constraint like "Father≡Man⊓∃hasChild.Person, hasChild(mdg, mj), Person(mj), Man(mdg)" can be formalized using Protege. A caution message is thrown after running Jess stating that "mdg now an individual of the class Father".

In the third step the classical we considered the often cited wine ontology[20]. There are 1418 facts transformed from XSLT, and 5840 assertions inferred from Jess, while running XSLT in 1 second and Jess in 29 second. Besides rdf:type and rdfs:subClassOf, user-defined properties like "hasMaker" or "locatedIn" are the source of new Jess assertions.

Finally, a larger ontology with approximately 1000 concepts from the medical domain is tested. Its source code has 33246 lines, and 20700 facts appear after XSLT, however the inferred 74238 assertions are almost all about rdf:type and rdfs:subClassOf since the ontology mainly consists of TBox assertions. The XSLT engine needed 1 second for the syntactical transformation, while running the Jess rule engine took approximately 300 seconds.

6.2 Related work

Related work towards the integration of rules and ontologies in the Semantic Web can be roughly divided into two categories: (1) extension approaches, which directly extend OWL knowledge bases with rules, such as SWRL[HPSB$^+$04], a combination of the OWL with the Unary/Binary Datalog Rule Markup Language, and the latest SWRL FOL[PS04] proposed to include an axiom for arbitrary first-order formula; (2) limitation approaches, which focus on the fragment of OWL, such as DLP(Description Logic Programs)[GHVD03], an intermediate

[20] http://www.w3.org/TR/owl-guide/wine.rdf

KR contained within the intersection of DL and LP, or OWL Lite$^-$[dBPF04], a strict subset of OWL Lite which can be translated into Datalog.

In the first category a SWRL Tab Widget[GI04] has been developed, which combines SWRL rules and OWL ontologies with the Protege OWL Plugin, Jess and Racer. Similarly, SweetJess[GGF02] (Semantic WEb Enabling Technlogies for Jess) defines a "DamlRuleML" ontology to deal with the Courteous Logic Programs. ROWL[SGS04] (Rule Language in OWL and Translation Engine for JESS) also uses a specially developed ontology to embody rules. As mentioned among these approaches, a specified namespace is required, like "swrl:","damlRuleML:" or "rowl:", to declare the elements of a rule such as head, body, variable, atom and so on. Nothing that, no special namespace is provided in our model but we closely follow up in spirit to extension. That is, all classes, properties and individuals are handed over from the OWL ontology and are expressed directly in rules. Furthermore the combination of properties could also be defined as a new rule to add more expressive power. On the other hand, as to the knowledge of the authors, the mentioned systems do not cover the RDF/OWL semantics to a satisfactory extent, ignoring owl:Restrictions or boolean expressions, which are emphasized in our model.

The second class of approaches concentrates on isolating proper subsets of OWL so as to be translated directly and completely without any loss of semantics. However, our intension was to allow OWL ontologies to be extended with rules, which means that we focus on scenarios where one needs to represent facts beyond the OWL expressive power.

A similar work was presented in OWLJessKB[21] which is a description logic reasoner for OWL, whose semantics is implemented in Jess. OWLJessKB makes use of the stand-alone distribution of ARP2 packaged with Jena2 to parse RDF syntax, resulting in a huge libraries imported, while in our work a simple XSLT sheetstyle is enough. Moreover, the semantic transformation included in OWLJessKB considers less entailment rules than in our case. Two error checking (misusing as rdfs:Class and as rdfs:Property), as well as only two warning messages (omitting as rdfs:domain and rdfs:range) are generated for further changes in the ontology.

6.3 Future work

Less than 100 lines in Java, we implement the transformation process of OWL files with XSLT and draw inferences of OWL ontology and individuals with Jess. All expressive restrictions are handled with the help of error or caution messages, and the inferred assertions are helpful for the author to recognize possible extensions of the ontology.

We are aware of the high space complexity required by the Rete algorithm used in Jess, but we note that the time complexity could be linear. To tackle this problem we can consider using a persistent storage system. In order for Semantic Web to work in real-world scenarios outside of the research lab, it must be possible to reason with scalable collections, for example by using database techniques as in InstanceStore. We plan to investigate this issue in the future in order to develop a tool taking advantages of the three technologies. Building ontologies including classification and consistency checking can be realized ef-

[21] http://edge.cs.drexel.edu/assemblies/software/owljesskb/

ficiently using Description Logics. Reasoning over large sets of individuals can be performed preferentially using logic programming systems, while retrieving and storing information persistently can rely on databases.

References

[BG94] Chitta Baral and Michael Gelfond. Logic Programming and Knowledge Representation. *Journal of Logic Programming*, 1994.

[dBPF04] Jos de Bruijn, Axel Polleres, and Dieter Fensel. OWL Light. *Available at http://www.wsmo.org/2004/d20/v0.1/20040629/*, 2004.

[Eri03] Henrik Eriksson. Using JessTab to Integrate Protege and Jess. *IEEE Intelligent Systems*, 18(2):43–50, 2003.

[For82] Charles L. Forgy. Rete: A Fast Algorithm for the Many Pattern Many Object Pattern Match Problem. *Artificial Intelligence*, 19(1):17–37, 1982.

[GGF02] Benjamin N. Grosof, Mahesh D. Gandhe, and Timothy W. Finin. SweetJess: Translating DamlRuleML to Jess. In *Rule Markup Languages for Business Rules on the Semantic Web, June 2002, Sardinia, Italy*, 2002.

[GHVD03] Benjamin Grosof, Ian Horrocks, Raphael Volz, and Stefan Decker. Description Logic Programs: Combining Logic Programs with Description Logics. In *WWW 2003, Budapest, Hungary, May 2003*, 2003.

[GI04] Christine Golbreich and Atsutoshi Imai. Combining SWRL rules and OWL ontologies with Protege OWL Plugin, Jess, and Racer. *7th International Protégé Conference, Bethesda, Maryland*, July 2004.

[HBD04] David Hirtle, Harold Boley, and Mike Dean. SWRL RuleML Accessing SWRL Properties as Foreign Atoms. *Available at http://www.ruleml.org/swrl/*, 2004.

[HM04] Patrick Hayes and Brian McBride. RDF Semantics. *Available at http://www.w3.org/TR/rdf-mt/*, 2004.

[HPS04] Ian Horrocks and Peter F. Patel-Schneider. A Proposal for an OWL Rules Language. In *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. *Available at http://www.w3.org/Submission/SWRL/*, 2004.

[HPSvH03] Ian Horrocks, Peter F. Patel-Schneider, and Frank van Harmelen. From SHIQ and RDF to OWL: The making of a web ontology language. *Journal of Web Semantics*, 1(1):7–26, 2003.

[KLW95] Michael Kifer, Georg Lausen, and James Wu. Logical Foundations of Object-Oriented and Frame-Based Languages. *Journal of ACM*, May 1995.

[KR03] Joseph Kopena and William Regli. DAMLJessKB: A Tool for Reasoning with the Semantic Web. *IEEE Intelligent Systems*, 18(3):74–77, May 2003.

[MLYL04] Jing Mei, Shengping Liu, Anbu Yue, and Zuoquan Lin. An Extension to OWL with General Rules. In *Rules and Rule Markup Languages for the Semantic Web: Third International Workshop, RuleML 2004, Hiroshima, Japan, November 8, 2004. Proceedings*, pages 155–169. Springer-Verlag Heidelberg, 2004.

[PS04] Peter F. Patel-Schneider. A Proposal for a SWRL Extension to First-Order Logic. *Available at http://www.daml.org/2004/11/fol/proposal*, 2004.

[PSHH04] Peter F. Patel-Schneider, Patrick Hayes, and Ian Horrocks. OWL Web Ontology Language Semantics and Abstract Syntax. *Available at http://www.w3.org/TR/owl-absyn/*, 2004.

[SGS04] Norman Sadeh, Fabien Gandon, and Mithun Sheshagiri. ROWL: Rule Language in OWL and Translation Engine for JESS. *Developers Day at the 2004 World Wide Web Conference*, May 2004.