Running head: Rule-based OWL Reasoning Systems

Rule-based OWL Reasoning Systems: Implementations, Strengths and Weaknesses

Georgios Meditskos and Nick Bassiliades

Department of Informatics, Aristotle University of Thessaloniki, Greece

ABSTRACT

This chapter is focused on the basic principles behind the utilization of rules in order to perform reasoning about the Web Ontology Language (OWL), a Description Logic-based language that is the W3C recommendation for creating and sharing ontologies in the Semantic Web. More precisely, we elaborate on the entailment-based OWL reasoning (EBOR) paradigm, which is based on the utilization of RDF/RDFS and OWL entailment rules that run on a rule engine, applying the formal semantics of the ontology language. To this end, seven EBOR systems are described and compared, analyzing the different approaches. Despite the closed rule environment, which comes in contrast with the open nature of the Semantic Web, and the fact that OWL semantics are partially mapped into rules, the rule-based OWL reasoning paradigm can give great potentials in the Semantic Web, enabling the utilization of rule engines on top of ontology information.

Rule-based OWL Reasoning Systems: Implementations, Strengths and Weaknesses

INTRODUCTION

Rule-based systems have been extensively used in several applications and domains, such as e-commerce, personalization, games, businesses and academia. They offer a simplistic model for knowledge representation for both domain experts and programmers; experts usually find it easier to express knowledge in a rule-like format and programmers usually find rule-based programming easier to understand and manipulate, decoupling computation from control. The first is performed by the rules whereas the latter is determined by the rule engine itself, that is when and how to apply the rules. In that way, it is more easily to add new rules or data, especially in continuously changing environments.

Nowadays, the Web has been evolved in a large repository of information and has become a useful means of communication and knowledge sharing. However, in order to exploit the Web to its full extent, information should become understandable not only to humans but also to machines. Towards this need, the Semantic Web initiative (W3C, 2008) works on standards, technologies and tools in order to give to the information a well-defined meaning, enabling computers and people to work in better cooperation. It is also worth mentioning the effort to design and build semantic Web services (Paolucci & Sycara, 2003), that are semantically annotated Web services using service description standards based on ontologies (OWL-S, 2004; Roman et al., 2005). Ontologies are considered as a primary key for the Semantic Web since they provide a controlled vocabulary of concepts, each with explicitly defined and machine processable semantics. The Web Ontology Language (OWL) (McGuinness & Harmelen, 2004) is the W3C recommendation for creating and sharing ontologies on the Web. It provides the means for ontology definition and specifies formal semantics on how to derive new information.

There are mainly two modeling paradigms for the Semantic Web. The first paradigm is based on the notion of the Classical Logics, such as the Description Logics (Baader, 2003) on which the OWL is based. In this case, the semantics of OWL ontologies can be handled by DL reasoning systems, such as Pellet (Sirin, Parsia, Grau, Kalyanpur & Katz, 2007), RacerPro (Haarslev & Moller, 2003) and Fact++ (Tsarkov & Horrocks, 2006) that reuse existing DL algorithms, such as tableaux-based algorithms (Baader & Sattler, 2001). The other paradigm is based on the Datalog paradigm. In this case, a subset of the OWL semantics is transformed into rules that are used by a rule engine in order to infer implicit knowledge. There are major differences between these two paradigms, including computational and expressiveness aspects. For example, the DL reasoning engines have a rather inefficient instance reasoning performance, whereas rules are insufficient to model certain situations related to the open nature of the Semantic Web. Obviously, the selection of the most suitable modeling paradigm depends on the domain and the needs of the application.

This chapter is focused mainly on the practical aspects of the implementation of a rule-based OWL reasoning system using OWL entailment rules (Horst, 2005), describing the way a rule engine can be used in order to reason about OWL ontologies. After a short background about the Semantic Web, the OWL language and the basic approaches behind the combination of rules and ontologies, a description of the basic foundations of the EBOR paradigm is given, explaining the way the entailment rules can operate over ontological data in order to apply semantic relationships. Furthermore, the benefits and limitations are discussed between the

approach of building a rule-based OWL reasoning system based on a general-purpose rule engine and developing from scratch an OWL-aware rule engine. To this end, seven existing EBOR systems are described and compared that follow different implementation directions.

The chapter presents also the basic arguments of the debate about the suitability of the Classical and the Datalog paradigms for the Semantic Web. Notions such as the open and closed-world semantics, the unique name assumption and the reasoning complexity are addressed for each modeling paradigm, highlighting the basic differences.

BACKGROUND

Semantic Web and the Web Ontology Language

Today's Web is suitable for human consumption and is organized around content presentation and not information meaning. The Semantic Web vision (Berners-Lee, Hendler & Lassila, 2001) emerged after Information Retrieval received great attention in the late 90s and the term metadata was coined. Metadata is often described as "data about data" and is used to facilitate the understanding, use and management of other data.

Machines should be able to reason over the represented data, drawing conclusions that seem obvious to humans but not to machines. Reasoning and logic have been under extensive study in AI and their ultimate goal is to make implicit knowledge explicit. Using automated reasoning in the Semantic Web can help uncover implicit knowledge hidden into metadata.

The Semantic Web initiative (W3C, 2008) tries to solve problems related to knowledge representation by suggesting standards, tools and languages for information annotation. Semantic Web can be considered as an extension of the current Web where information has unambiguous and well-defined meaning, enabling machines/agents to understand the semantics of the information and not only base on the syntax. Ontologies play a key role to the evolution of the Semantic Web and are widely used to represent knowledge by describing data in a formal and explicit way.

The Web Service Modeling Language (WSML) (de Bruijn, Lausen, Polleres, & Fensel, 2006) defines a syntax and semantics for ontology descriptions. The goal of the development of WSML is to investigate the usage of different formalisms, most notably Description Logics (DLs) and Logic Programming (LP), in the context of Ontologies and Web services. There are five variants of WSML, namely WSML-Core, WSML-DL, WSML-Flight, WSML-Rule and WSML-Full, with different logical expressiveness and underlying language paradigms, allowing users to choose between expressiveness and complexity. WSML is based on the conceptual model of WSMO (Roman et al., 2005).

The Web Ontology Language (OWL) (McGuinness & Harmelen, 2004) is the W3C recommendation for creating and sharing ontologies in the Web and its theoretical background is based on the DL (Baader, 2003) knowledge representation formalism, a subset of predicate logic. It has been emerged as the solution to the expressive limitations of RDF and RDF Schema (RDFS) (Hayes, 2004) that offer the possibility to define only simple hierarchical relationships among concepts and properties, domain and range property restrictions and instances of concepts. OWL is a richer vocabulary description language for describing properties and classes, such as relations between classes (e.g., disjointness), cardinality (e.g. "exactly one"), equality, richer typing of properties, characteristics of properties (e.g., symmetry), and enumerated classes (Antoniou & Harmelen, 2004).

The formal semantics of the OWL language enable the application of reasoning techniques in order to make logical derivations, involving class membership, equivalent classes, ontology consistency, and instance classification. These derivations are performed by the reasoners, which are systems able to handle and apply the semantics of the ontology language. A general reasoning procedure is depicted in Figure 1 and involves two phases, namely the mapping phase of the asserted knowledge into a knowledge representation formalism and the application of an inference mechanism in order to perform the basic derivations.

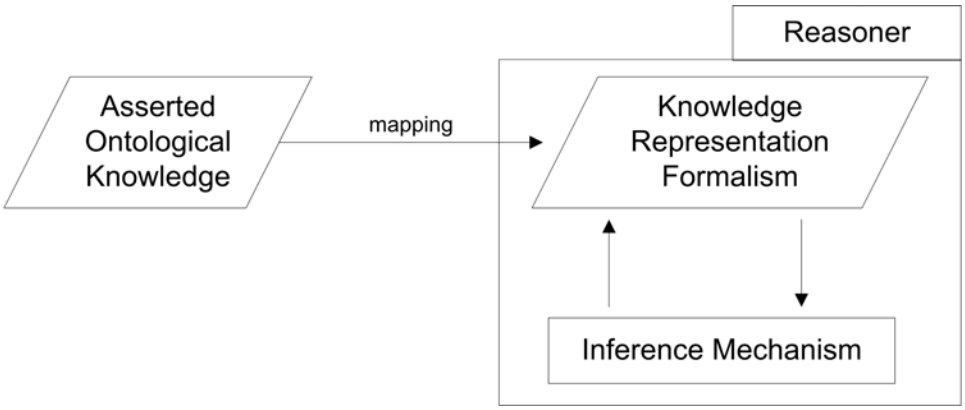

Figure 1. The abstract architecture of an OWL reasoner.

An OWL ontology is actually a finite set of DL axioms, such as axioms about concepts, concept inclusions ($C \sqsubseteq D$), role definitions, role inclusions ($R \sqsubseteq S$), concept assertions ($C(a)$) and role assertions ($R(a, b)$), where C, D are concepts, R, S are roles and a, b are instances. These axioms can be divided into two categories, namely the TBox and the ABox of the ontology (Baader, 2003). The TBox consists of the concept and role definitions/inclusions, and the ABox of concept and role assertions. Intuitively, the TBox refers to the schema of the ontology, whereas the ABox to the instances.

The ability to extract new information is a critical characteristic of every reasoning system and defines its reasoning completeness and soundness. Unfortunately, there is a tradeoff between scalability, in terms of reasoning performance, and ontology language expressiveness. The more expressive is the language the less efficient is the reasoning performance. For that reason, OWL comes in three flavors, namely OWL Lite (SHIN(D)), OWL DL (SHOIQ(D)) and OWL Full, having different expressiveness. This is achieved by restricting the available constructs that can be used or by restricting the way that each construct can be used during the modeling of a domain. OWL Full is syntactically and semantically upward-compatible with RDF, allowing the use of all the OWL languages primitives. It also allows the combination of these primitives in arbitrary ways with RDF and RDF Schema. However, this great degree of expressiveness does not offer computational guarantees. Thus, most reasoning systems target at the OWL DL and OWL Lite sublanguages. OWL DL restricts the way the constructors from OWL and RDF may be used, whereas OWL Lite further restricts the language constructors of OWL DL (Antoniou & Harmelen, 2004; McGuinness & Harmelen, 2004). In terms of the WSML variants, OWL Lite is a semantic superset of WSML-Core, whereas OWL DL is semantically equivalent to WSML-DL.

OWL is built upon RDF and RDFS and has the same syntax, the XML-based RDF syntax (Beckett & McBride, 2004). We present as an example the region ontology that describes the concept Region, a transitive property subRegionOf and three instances with specific subRegionOf relationships.

```
<owl:Class rdf:ID = "Region" />
<owl:TransitiveProperty rdf:ID = "subRegionOf" >
        <rdfs:domain rdf:resource="#Region" />
        <rdfs:range rdf:resource="#Region" />
</owl:TransitiveProperty>
<Region rdf:ID = "region1" />
<Region rdf:ID = "region2" >
        <subRegionOf rdf:resource="#region1" />
</Region>
<Region rdf:ID = "region3">
        <subRegionOf rdf:resource="#region2" />
</Region>
```

A more machine processable syntax is the N-Triples format (Grant & Beckett, 2004), that is a textual format for RDF graphs which stems directly from the RDF/XML syntax. More specifically, N-Triples is a line-oriented format where each triple must be written on a separate line, and consists of a subject, a predicate, and an object, followed by a period. For example, the region ontology can be represented in the N-Triple format with nine triples as:

```
t1: <Region> <rdf:type> <owl:Class> .
t2: <subRegionOf> <rdf:type> <owl:TransitiveProperty> .
t3: <subRegionOf> <rdfs:domain> <Region> .
t4: <subRegionOf> <rdfs:range> <Region> .
t5: <region1> <rdf:type> <Region> .
t6: <region2> <rdf:type> <Region> .
t7: <region3> <rdf:type> <Region> .
t8: <region2> <subRegionOf> <region1> .
t9: <region3> <subRegionOf> <region2> .
```

OWL Ontologies and Rules

The development of Semantic Web proceeds in layers where each layer is built on top of the others (Berners-Lee et al., 2001). Currently, the ontology layer has reached a sufficient level of maturity, having OWL as the basic form for ontology definition. The next step is to move on the higher levels of unifying logic and proof, which are built on top of the ontology layer. In the latest version of the Semantic Web stack, rules lay next to the ontology layer and they are considered as the primary key, since (a) they can serve as extensions of, or alternatives to, DL based ontology languages and (b) they can be used to develop declarative systems using ontological information.

The Benefits of Combining DLs and Rules

Although there is a lot of debate about the suitability of Logic Programming (LP) in the domain of the Semantic Web, many research efforts have been focused on the *mapping*, *intersection* or *combination* of DLs and LP in order to overcome the shortcomings that emerged

during the development of practical OWL applications (Patel-Schneider & Horrocks, 2006; Motik, Horrocks, Rosati & Sattler, 2006). Such approaches are important for many aspects of the Semantic Web, such as

- **Querying:** It is interesting to consider combining DLs with the rule paradigm in order to state expressive instance queries, since DL reasoning engines have rather a low ABox reasoning and querying performance (Haarslev & Moller, 1999; Horrocks, Li, Turi & Bechhofer, 2004a; Hitzler & Vrandecic, 2005).
- **Non-monotonicity:** DLs follow the principle of the open-world assumption (monotonic). However, sometimes it is preferable to introduce non-monotonicity in the DLs (Motik et al., 2006), e.g. the notion of negation as failure in logic programs. In that way, it is possible to incorporate the semantics of closed-world reasoning over DL knowledge bases that are used in relational databases.
- **DLs' expressivity:** Rules can serve as extensions of description logic based ontology languages (Horrocks & Sattler, 2004c; Horrocks, Kutz & Sattler, 2006), allowing the definition of richer semantic relationships.
- **Integrity constraints:** Sometimes is useful to be able to define integrity constraints, i.e. constraints over the ABox of the ontology. For example, the ontology that contains the axioms $\mathsf{Person} \sqsubseteq \mathsf{hasSSN.SSN}$ and $\mathsf{Person(george)}$ is satisfiable in OWL, even if we do not define an SSN number for george (open-world semantics). By introducing integrity constraints, we are able to treat such cases as checks, rather than deriving new information (Motik, Horrocks, & Sattler, 2007).

Intersection of DLs and Logic Programming

Grosof, Horrocks, Volz and Decker (2003) define the intersection of LP and DL, namely the Description Logic Programs (DLP). Actually, DLP is the most expressive sublanguage of OWL DL that can be efficiently mapped to Datalog and it is simpler than OWL Lite. In that way, it is possible to interoperate between rules and ontologies, transforming LP to DL and vice versa.

WSML-Core corresponds with the intersection of DL and Horn Logic (DLP) (without function symbols and without equality), extended with datatype support in order to be useful in practical applications. WSML-Core is fully compliant with a subset of OWL.

While DLP is the intersection of LP and DL, the OWL Flight (de Bruijn, Lara, Polleres & Fensel, 2005) is an ontology language based totally on the LP subset of OWL. It is inspired by DLP and imposes certain extensions in the area of datatypes, database-style constraints, such as cardinality and value constraints, and meta-modeling. OWL Flight restricts the OWL syntax such that it falls in the Datalog fragment and thus query answering can be done using an LP implementation. WSML-Flight and WSML-Rule are based on the LP paradigm, rather than the DL paradigm. They allow non-monotonic negation but do not allow classical negation, full disjunction and existential quantification.

Mapping of DLs on Logic Programming

Van Belleghem, Denecker and De Schreye (1997) present a mapping of DL knowledge bases in ALCN to open logic programs, exploring the computational correspondences between a typical algorithm for DL inference and the resolution procedure for open logic programs. Baral

(2003) and Swift (2004) reduce inference in the DL ALCQI to query answering from the answer sets of logic programs. Horst (2004) defines the pD* semantics (entailments) as a weakened variant of OWL Full and then (Horst, 2005) the pD* semantics were extended to apply to a larger subset of the OWL vocabulary. Meditskos and Bassiliades (2008a) combine a DL reasoner with a dynamic implementation of entailment rules in order to increase scalability. Reduction (Hustadt, Motik & Sattler, 2004; Motik et al., 2006) is also a way of building a reasoning system reducing a DL KB to disjunctive Datalog programs.

Combing DL and Logic Programming

The major flaw of the mapping approaches is the fact that there is not an unrestricted mapping of OWL semantics into the rule paradigm, and thus the resulting languages have restricted semantics, handling a subset of OWL DL. To solve this expressivity problem, many research efforts have been focused on the combination of DL and LP. Such a combination is realized following either a *hybrid* or a *homogeneous* approach (Antoniou et al., 2005).

Hybrid approach: The hybrid combination follows a modular architecture of two subsystems, each of which deals with a distinct portion of the knowledge base. More specifically, it combines the reasoning capabilities of a DL reasoner and the rule execution capabilities of a rule engine in order to define rules on top of the ontological information. Rule and ontology predicates are strictly separated and the ontology predicates can be used as constraints in rules. The hybrid approaches can be further classified into bidirectional and unidirectional, according to whether the derived knowledge flows from the rule module to the DL module or not. In the former case, DL constraints can be used in the head of the rules and thus, the ontological knowledge is altered, allowing the development of ontologies on top of rules (Wang, Billington, Blee & Antoniou, 2004; Rosati, 2006; Kattenstroth, May & Schenk, 2007). In the latter case, the information flows only from the DL component to the rule component by allowing only rule predicates to be used in rule bodies and thus the ontological information remains unchanged (Donini, Lenzerini, Nardi & Schaerf, 1998; Levy & Rousset, 1998; Rosati, 1999; Eiter, Lukasiewicz, Schindlauer & Tompits, 2004; Rosati, 2005; Drabent, Henriksson & Maluszynski, 2007).

Homogeneous approach: The homogeneous approaches treat rule and ontology predicates homogeneously, as a new single logic language. The general idea is that the rules can use unary and binary predicates from the ontology (i.e., classes and properties) as well as predicates that occur only in rules (rules predicates). In order to maintain the decidability of the integrated language, there is usually a safety condition that restricts variables occurring in the head of a rule to those that occur in at least one positive rule predicate in the body of the rule. Intuitively, in homogeneous approaches, the OWL semantics are mapped into a rule-based formalism, e.g. Datalog rules that coexist in the KB with rule predicates, enhancing the expressivity. The homogeneous approaches can be used either for building rule programs on top of ontologies or ontologies on top of rules. Thus, a new reasoner is needed, able to handle the new homogeneous language that emerges (Heymans, Predoiu, Feier, de Bruijn & Nieuwenborgh, 2006; Mei, Lin & Boley, 2007; de Bruijn, Eiter, & Tompits, 2008). In fact, the mapping approaches we described previously can be considered as the first step for building a homogeneous system (Horst, 2005; Motik et al., 2006).

Another proposal is the Semantic Web Rule Language (SWRL) (Horrocks et al., 2004b), a non-safe approach to the integration of rules and DLs in which rules are interpreted under the classical first order logic semantics. The addition of this kind of rules to DLs leads to undecidability of reasoning.

In the following section we describe the basic principles of the entailment-based OWL reasoning (EBOR) paradigm that enables the materialization of OWL semantics into the KB of a rule engine using OWL entailment (inference) rules. It is actually based on the pD* semantics where the DL knowledge base is mapped on a rule engine and the entailment rules are encoded in the rule engine's language. The EBOR paradigm can be considered as the first step in realizing a homogeneous combination of OWL and rules in order to build rule programs on top of ontologies, as well as ontologies on top of rule programs, since the rule program coexists with the inference rules in the rule base, and thus, the rule execution is interleaved with the inference procedure. Note that the EBOR paradigm by itself does not really bring many of the benefits of adding rules to ontologies, such as integrity constraints, since it is a subset of OWL and it is based on open-world semantics. However, by adding custom rules on the underlying rule engine, we are able to fully exploit the benefits that stem from the combination of rules and ontologies.

ENTAILMENT-BASED OWL REASONING

The EBOR paradigm is realized by using a rule engine as the inference engine in the architecture of Figure 1, implementing RDF/OWL entailment rules (Hayes, 2004; Horst, 2005).

The semantics of RDF and RDFS can be captured using entailments rules (Hayes, 2004), which are rules that denote the information that should be derived based on existing one. Intuitively, an entailment rule is an if-then rule that denotes the knowledge that should be inferred (rule head) based on existing knowledge (rule body). The body consists of RDF statements, where variables can occupy any of the three possible positions in the triple (that of a subject, of a predicate, or of an object). The head of the rule comprises of one or more consequences, each of which represents in its turn an RDF statement. The consequences may not contain free variables, i.e. such that are not used within the body of the rule. The list of the RDF/RDFS entailments is defined in Hayes (2004). We give as an example the *rdfs9* entailment rule using the N-Triple notation.

```
if      <c> <rdfs:subClassOf> <d> . ∧
        <x> <rdf:type> <c> .
then    <x> <rdf:type> <d> .
```

The *rdfs9* entailment rule actually defines the subsumption characteristic of the rdfs:subClassOf property: if there is an instance <x> defined to belong to the class <c>, and <c> is defined as a subclass of the class <d>, then <x> is also of type <d>.

Although there is a complete set of RDF and RDFS entailments (Hayes, 2004), such a complete set does not exist for OWL due to the great degree of expressiveness. Horst (2004) defines the pD* semantics as a weakened variant of OWL Full and then in (Horst, 2005) the pD* semantics were extended to apply to a larger subset of the OWL vocabulary, which includes FunctionalProperty, InverseFunctionalProperty, sameAs, SymmetricProperty, TransitiveProperty, inverseOf, equivalentClass, equivalentProperty, hasValue, someValuesFrom, allValuesFrom, differentFrom and disjointWith. More precisely, the pD* semantics can be realized by 23 entailment rules and 2 inconsistency rules. To exemplify, we present the *rdfp4* entailment that handles the

values of transitive properties that are defined using the owl:TransitiveProperty OWL construct (Antoniou & Harmelen, 2004).

```
if      <p> <rdfs:type> <owl:TransitiveProperty> . ∧
        <s> <p> <x> . ∧
        <x> <p> <z> .
then    <s> <p> <z>.
```

The pD* semantics extend RDFS and they are defined in a way analogous to the if-semantics of RDFS, leading to simple entailment rules that can be used to extend RDF reasoners. In other words, we do not obtain the full power of OWL's iff semantics in pD* semantics. Instead, they represent a reasonable interpretation that is useful for drawing conclusions about instances in an ontology and that lead to simple entailment rules with a relatively low computational complexity (consistency is in P and entailment is NP-complete, and in P if there are not blank nodes in the target graph). The pD* semantics are going to be standardized (at the time this chapter was being written) as the OWL RL profile of OWL 2 (Motik et al., 2008).

In the EBOR paradigm, the asserted knowledge, that is the knowledge that stems directly from the ontology definition, is mapped into an internal rule engine representation format, and inference rules, which are expressed in the language of the rule engine, are applied in order to deduce new knowledge or to check the consistency of the ontology, based on OWL entailments. To exemplify, let *S* be the set of triples of an ontology, where *S* = {<A subClassOf B>, <x type A>}. By implementing the *rdfs9* entailment rule, we get that *S* = {<A subClassOf B>, <x type A>, <x type B>}.

Therefore, for the development of an EBOR system, three issues should be tackled:

- **Ontology mapping:** An EBOR system should define a mapping procedure of the ontological knowledge into the KB of the rule engine that uses. Usually, such a mapping procedure is performed over the ontology triples. The purpose of this phase is to generate an internal, rule engine-specific representation of the ontological information where the entailment rules will be applied on.
- **Inferencing process:** An EBOR system should implement the desirable number of entailment rules expressed in the engine's rule language. This phase actually defines the reasoning completeness of the EBOR system that usually comes with implementations of different expressiveness according to the number of entailments that are implemented.
- **Query support:** An EBOR system should be able to answer queries about the semantic derivations of its KB. Since the core system is a rule engine, the query infrastructure is implemented with query rules. These query rules follow either the rule language of the underlying rule engine, or they have a standard-based syntax (Wagner, Antoniou, Tabet & Boley, 2004; Prud'hommeaux & Seaborne, 2008).

There are two approaches for the development of an EBOR system, namely the *extended* and the *native* approach. An extended entailment-based OWL reasoning (E-EBOR) system is built on top of an existing, general purpose rule engine that augments it with the ability of manipulating ontological information. This incorporates the ability of transforming the ontological information into facts and populating its rule base with the appropriate inference rules. A native entailment-based OWL reasoning (N-EBOR) system is built from scratch and draws conclusions directly on the OWL data model. Each approach has advantages and

disadvantages and the right choice depends on the requirements of the application. More specifically:

- **Reasoning Performance:** Since an N-EBOR system is built directly upon the OWL data model, it has increased reasoning performance, as far as speed issues are concerned, comparing it to the E-EBOR paradigm that does not apply any optimization in the way it handles the ontological information. Thus, an N-EBOR system is an appropriate choice in the cases where reasoning speed is a critical requirement.
- **Ontology Utilization:** The use of an E-EBOR system gives the opportunity to efficiently utilize the ontology information by building rule-based applications. Ontologies can be inserted into the system and, after the application of the inference rules, user-defined rules can operate over the inferred knowledge. In that way, an EBOR system built on top of a general purpose rule engine gives the opportunity to reuse the practicality, efficiency and optimized techniques that rule engines have obtained throughout the years of their development. On the other hand, an N-EBOR system built from scratch tends to throw away decades of research and development on efficient and robust rule engines. Therefore, an E-EBOR has increased capabilities concerning post-reasoning utilization of ontological information into rule programs.

In the following sections we present in detail examples of E-EBOR and N-EBOR systems, describing the way they map ontologies into their KBs, as well as the rule language that use in order to implement the inference rules and to query the KB. For the legibility of the description, we present also the way each system manipulates the region ontology we presented in the beginning of the chapter.

Extended Entailment-based OWL Reasoning Systems

In this section we present three E-EBOR systems, namely OWLJessKB (Kopena, Regli, 2003), F-OWL (Zou, Finin & Chen, 2004), and O-DEVICE (Meditskos & Bassiliades, 2008b).

OWLJessKB

OWLJessKB is built on top of the Jess (Friedman-Hill, 2003) production rule engine. Jess is an Expert System Shell developed in the Java programming language. It uses an enhanced version of the Rete algorithm (Forgy, 1982) to process rules. Jess has many unique features including backwards chaining and working memory queries, and it can directly manipulate and reason about Java objects.

The functionality of OWLJessKB involves the translation of OWL ontologies into RDF triples and their transformation into Jess facts in order to build the KB on where the entailment rules will be applied. More specifically, each RDF triple of the form <subject> <predicate> <object> is transformed into a fact of the following Jess template construct:

```
(deftemplate triple (slot predicate) (slot subject) (slot object)).
```

In that way, an OWL ontology is represented in Jess as a set of template facts and OWL semantics are implemented as Jess production rules over the KB. To exemplify, the triples of the region ontology are transformed into the following Jess facts:

```
j1: (triple (predicate "rdf:type") (subject "Region") (object "owl:Class")).
j2: (triple (predicate "rdf:type") (subject "subRegionOf") (object "owl:TransitiveProperty")).
j3: (triple (predicate "rdfs:domain") (subject "subRegionOf") (object "Region")).
j4: (triple (predicate "rdfs:range") (subject "subRegionOf") (object "Region")).
j5: (triple (predicate "rdf:type") (subject "region1") (object "Region")).
j6: (triple (predicate "rdf:type") (subject "region2") (object "Region")).
j7: (triple (predicate "rdf:type") (subject "region3") (object "Region")).
j8: (triple (predicate "subRegionOf") (subject "region2") (object "region1")).
j9: (triple (predicate "subRegionOf") (subject "region3") (object "region2")).
```

The entailment rules of OWLJessKB are actually Jess production rules that match the facts that exist in the KB and deduce the implicit knowledge in the form of new facts. To exemplify on the syntax of the entailments, we present the production rule for the manipulation of the values of the transitive properties (*rdfp4* entailment in (Horst, 2005)).

```
(defrule transitive-property
        (triple (predicate "rdf:type") (subject ?prop) (object "owl:TransitiveProperty"))
        (triple (predicate ?prop) (subject ?x) (object ?y))
        (triple (predicate ?prop) (subject ?y) (object ?z))
=>
        (assert (triple (predicate ?prop) (subject ?x) (object ?z)))
)
```

The application of the transitive rule over the facts of the region ontology results in the assertion of the following inferred fact into the KB.

```
j10: (triple (predicate "subRegionOf") (subject "region3") (object "region1")).
```

The query infrastructure of OWLJessKB is based on the defquery construct of Jess, which is a special kind of rule with no right-hand-side (RHS). A query is actually a pattern that is used to search the working memory and the matched facts are returned in a list. In the region ontology, we can retrieve all the instances of the Region concept that have in their subRegionOf property the instance value region1 as

```
(defquery region-instances
        (triple (subject ?s) (predicate "rdf:type") (object "Region"))
        (triple (subject ?s) (predicate "subRegionOf") (object "region1"))
)
```

The example query matches both the j8 and j10 facts which can be retrieved using the OWLJessKB Java API.

F-OWL

F-OWL is built on top of Flora2 (Yang, Kifer & Zhao, 2003), an implementation of the F-Logic language (Kifer, Lausen & Wu, 1995) for data definition and querying that makes use of the XSB Logic Programming and Deductive Database system (Sagonas, Swift & Warren, 1994). In contrast to OWLJessKB that transforms ontology triples directly into Jess facts, F-OWL

transforms ontology triples into the F-logic syntax, exploiting the semantics and the knowledge representation paradigm of a frame-based syntax.

F-Logic is a deductive, object-oriented knowledge representation language based on frames and extends first-order predicate calculus with:

- Objects with complex internal structure
- Class hierarchies and inheritance
- Typing
- Encapsulation

In this way, F-logic integrates the paradigms of logic programming and deductive databases with the object-oriented programming paradigm, and it 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. More specifically:

- **Ontology concepts:** F-logic supports the definition of concepts, as well as a hierarchy of concepts using subclass relationships, for example `A :: B` (`A` is subclass of `B`). In that way, the Region concept with the `subRegionOf` property of the region ontology can be represented as `Region[subRegionOf *=>> Region]`, which is called a signature. The `*=>>` symbol denotes a multivalue (`=>>` symbol), inheritable instance attribute (`*` symbol). Since F-logic is tightly connected to the object-oriented programming paradigm, the hierarchical relationships encapsulate property inheritance as well as class membership transitivity.
- **Ontology instances:** The definition of an instance in F-logic incorporates the declaration of the concept to where it belongs (`:` notation), as well as the values in the corresponding properties. In that way, the three instances of the region ontology can be represented as:

```
region1:Region.
region2:Region.
region3:Region.
region2[subRegionOf =>> region1].
region3[subRegionOf =>> region2].
```

F-OWL maps the ontology triples into Flora2 facts of the form `triple(s, p, o)` and applies a set of transformation rules that generate the F-logic code. For example, the ontology triples are transformed into Flora2 statements using the rule

```
S[P->>O] :- triple(S, P, O),
```

and the instance definitions are generated using the rule

```
A:B :-  A[rdf_type ->> B].
```

Obviously, the transformation procedure of ontological information into the internal rule engine representation format is more complicated in F-OWL than in OWLJessKB. However, F-OWL achieves a more compact and human-friendly representation of the ontological information than of OWLJessKB, exploiting object-oriented programming principles.

OWL semantics are implemented as Flora2 Prolog-like rules that operate over the generated KB. For example, the rule that implements the transitive entailment is defined as

```
S[P->>V] :- P[rdf_type->>owl_TransitiveProperty], S[P->>X], X[P->>V].
```

In that way, the following F-logic statement can be inferred in the region ontology, denoting that region3 is related to the region1 instance through the subRegionOf property.

```
region3[subRegionOf =>> region1].
```

Both the schema information associated with classes and the structure of individual objects can be queried by simply putting variables in the appropriate syntactic positions in the Prolog-like rules. Therefore, in order to retrieve all the instances of the Region concept with the region1 instance in their subRegionOf property, we should define the following query:

```
X : Region [subRegionOf ->> region1].
```

O-DEVICE

O-DEVICE is built on top of the CLIPS production rule engine (Riley, 1991). Its reasoning process is characterized by the transformation of ontological information into the object-oriented model of the COOL language of CLIPS and the application of inference production rules over the generated object-oriented schema. O-DEVICE has been also used in the domain of semantic Web service discovery and composition (Meditskos & Bassiliades, 2007).

CLIPS is a RETE-based production rule engine written in C that was developed in 1985 by NASA's Johnson Space Center and it has undergone continual refinement and improvement ever since. Today it is widely used throughout the government, industry and academia. One of the most interesting capabilities of CLIPS is that integrates the production rule paradigm with the object-oriented model, which can be defined using the COOL (CLIPS Object-Oriented Language) language of CLIPS. In that way, classes, attributes and objects can be matched on the production rule conditions (LHS), as well as to be altered on rules actions (RHS).

O-DEVICE transforms OWL ontologies into triples and applies a set of transformation rules in order to generate a COOL-based object-oriented schema of classes, attributes (slots) and objects. For example, the Region class and the subRegionOf property of the region ontology are transformed into a COOL defclass construct and the ontology instances into COOL objects, as follows:

```
(defclass Region
        (is-a owl:Thing)
        (subRegionOf (type INSTANCE-NAME)))

(make-instance region1 of Region)
(make-instance region2 of Region
        subRegionOf region1)
(make-instance region3 of Region
        subRegionOf region2)
```

Although this approach seems similar to the one followed by F-OWL, there is one major difference. The object-oriented model that is generated in O-DEVICE is fully compliant with object-oriented principles, whereas F-OWL is based only partially on object-oriented principles. To exemplify, consider the case where an ontology instance x is defined to belong to more than one classes, such as:

```
<x> <rdf:type> <Class1> .
<x> <rdf:type> <Class2> .
```

This case is represented directly in F-OWL, since Flora2 translates the F-logic statements into facts in XSB, using the frame-based syntax only for representing the information, as

```
x:Class1
x:Class2
```

However, since O-DEVICE is based on object-oriented principles, such a definition is not allowed, since every object can have only one class type. O-DEVICE handles this case by creating a subclass T of the two classes and defines x to be an instance of T, that is:

```
(defclass T
        (is-a Class1 Class2))

(make-instance x of T)
```

The reasoning procedure of O-DEVICE separates the TBox and the ABox reasoning activities, using static production rules for the former, in the same way as OWLJessKB and F-OWL, whereas it follows a template-based methodology for the latter, generating domain-dependent inference rules according to the degree of the expressiveness of the loaded ontology. To exemplify, the template rule for property transitivity (*rdfp4*) is defined in O-DEVICE as

```
(defrule <rule-name>
        (object (is-a <pd>) (name ?o1) (<p> $? ?o2 $?))
=>
        (bind $?v1 (send ?o1 get-<p>))
        (bind $?v2 (send ?o2 get-<p>))
        (send ?o1 put-<p> (union $?v1 $?v2)))
```

where <pd> denotes the domain of the transitive property <p>. O-DEVICE generates these rules at runtime, grounding the template (<>) elements with ontology TBox values. In that way, for *k* transitive properties in an ontology, *k* transitive rules will be generated. For the region ontology, one transitive inference rule for the subRegionOf will be generated as

```
(defrule subRegionOf
        (object (is-a Region) (name ?o1) (subRegionOf $? ?o2 $?))
=>
        (bind $?v1 (send ?o1 get-subRegionOf))
        (bind $?v2 (send ?o2 get-subRegionOf))
        (send ?o1 put-subRegionOf (union $?v1 $?v2)))
```

The deductive rule language of O-DEVICE supports querying over OWL instances represented as objects. The conclusions of deductive rules represent derived classes, whose objects are generated by evaluating these rules over the current set of objects. Each deductive rule is implemented as a CLIPS production rule that inserts a derived object when the condition of the deductive rule is satisfied. More specifically, the query for the instances of the Region class that have the region1 instance in their subRegionOf property is defined as

```
(deductiverule region-instances
        ?id<- (Region (subRegionOf $? [region1] $?))
=>
        (DERIVED (result ?id)))
```

The query denotes that the objects that match the LHS of the rule will be collected as property values in the result slot of the objects of the DERIVED class. Thus, the query will generate two objects of the DERIVED class with an arbitrary OID, with the region2 and region3 instances in their result slot.

Native Entailment-based OWL Reasoning Systems

In this section we present four EBOR systems that use rule engines built from scratch, namely Jena2 (McBride, 2001), Bossam (Minsu & Sohn, 2004), OWLIM (Kiryakov, Ognyanov & Manov, 2005) and BaseVISor (Matheus, Baclawski & Kokar, 2006).

Jena2

Jena2 is a programmatistic environment that contains a rule-based reasoner which is used to implement the RDFS and OWL entailment rules. Although the rule engine is also available as a general purpose rule engine, we classify Jena2 as an N-EBOR system, since the rule engine follows a triple-oriented syntax, mainly used for ontology reasoning. The reasoner supports rule-based inference over RDF graphs and provides forward chaining, backward chaining and a hybrid execution model. Actually, there are two internal rule engines: one forward chaining Rete engine and one tabled datalog engine. They can be run separately or the forward engine can be used to prime the backward engine which in turn will be used to answer queries.

The ontologies are transformed into triples, and inference rules are applied in order to materialize the semantics in the form of inferred triples. This approach is similar to the OWLJessKB system, rather than the frame-based approaches of F-OWL and O-DEVICE, with the exception that OWLJessKB is based on a general purpose rule engine (Jess) and not on a triple-oriented rule engine like the one that Jena2 uses over the RDF graphs.

In the forward chaining mode, any rules which fire and create additional triples, do so in an internal deductions graph and can in turn trigger additional rules. There is a remove primitive that can be used to remove triples and such removals can also trigger rules to fire in removal mode. This cascade of rule firings continues until no more rules can fire. When the inference procedure is completed, the inference graph will act as if it were the union of all the statements in the original model together with all the statements in the internal deductions graph generated by the rule firings. To exemplify on the forward chaining rule syntax of Jena, the transitive entailment rule is implemented as

```
[transProp-f: (?P rdf:type owl:TransitiveProperty), (?A ?P ?B), (?B ?P ?C) -> (?A ?P ?C)].
```

If the rule reasoner is run in backward chaining mode it uses an LP engine with a similar execution strategy to Prolog engines. When the inference model is queried, then the query is translated into a goal and the engine attempts to satisfy that goal by matching to any stored triples and by goal resolution against the backward chaining rules. This is also the way F-OWL performs inferencing based on Prolog-like rules. The backward chaining syntax of the transitive entailment in Jena is defined as

```
[transProp-b: (?A ?P ?C) <- (?P rdf:type owl:TransitiveProperty), (?A ?P ?B), (?B ?P ?C)].
```

The rule reasoner of Jena has the option of employing both the forward and backward rule engines in conjunction (hybrid mode). The forward engine runs and maintains a set of inferred statements in the deductions store. Any forward rules which assert new backward rules will instantiate those rules according to the forward variable bindings and pass the instantiated rules on to the backward engine. The hybrid rule syntax of the transitive entailment in Jena is defined as

```
[transProp-h: (?P rdf:type owl:TransitiveProperty) ->
    [transitiveProperty1b:  (?A ?P ?C) <- (?A ?P ?B), (?B ?P ?C)]]
```

By applying one of the three reasoners we have described in the region ontology, the triple (region3 subRegionOf region1) is inferred.

Queries are answered by using the backward chaining LP engine. Jena supports queries written in the SPARQL RDF Query Language (Prud'hommeaux & Seaborne, 2008), a W3C recommendation. The SPARQL query language consists of the syntax and semantics for asking and answering queries against RDF graphs. It contains capabilities for querying by triple patterns, conjunctions, disjunctions, and optional patterns. It also supports constraining queries by source RDF graph and extensible value testing. Results of SPARQL queries can be ordered, limited and offset in number, and presented in several different forms. To exemplify on SPARQL syntax, we present the query that retrieves all the instances of the Region class that have the region1 instance in their subRegionOf property.

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
SELECT ?x
WHERE
{
        ?x rdf:type Region.
        ?x subRegionOf region1
}
```

Bossam

Bossam is a Rete-based forward chaining N-EBOR engine, which is equipped with extended representational and extra-logical features:

- Support for both negation-as-failure and classical negation

- Relieved range-restrictedness in the rule heads
- Remote binding for cooperative inferencing among multiple rule engines

Following the EBOR paradigm, Bossam imports and translates OWL ontologies into a list of Bossam facts and then applies the OWL entailment rules expressed in the Buchingae rule language of Bossam. The translation process of Bossam involves the transformation of OWL documents into a collection of RDF triples and then each triple into a plain fact with three terms. In that way, Bossam translates RDF triples involved in declaring OWL classes and restrictions into 1-ary predicates, and the triples declaring property values into 2-ary predicates. Concerning the region ontology, the generated Bossam facts are depicted below:

```
owl:Class(Region);
owl:TransitiveProperty(subRegionOf)
rdfs:domain(subRegionOf, Region)
rdfs:range(subRegionOf, Region)
Region(region1);
Region(region2);
Region(region3);
subRegionOf(region2, region1);
subRegionOf(region3, region2);
```

A typical rule definition structure of Bossam's rule language is as follows:

```
rule ruleID is if condition-part then conclusion-part;
```

In the condition-part, the conditional elements are specified that should be true in order for the rule to fire. In the conclusion-part, some new knowledge elements are entailed. To exemplify, the transitive entailment rule is defined as

```
prefix owl = http://www.w3.org/2002/07/owl#;
rulebase transitiveProperty
{
        rule TransProp is
        if owl:TransitiveProperty(?p) and ?p(?x,?y) and ?p(?y,?z)
                then ?p(?x,?z);
}
```

However, the rule language of Bossam offers the opportunity to assert new rules in the conclusion part of a rule. The pattern at the consequent part is called the rule template. Thus, instead of deriving facts, Bossam generates rule instances from the rule templates. Therefore, Bossam has built-in the feature that O-DEVICE implements with templates. For example, the transitive rule can be defined also as

```
prefix owl = http://www.w3.org/2002/07/owl#;
rulebase transitivePropertyEx
{
        rule TransPropEx is
        if owl:TransitiveProperty(?p) then
                assert if ?p(?x,?y) and ?p(?y,?z) then ?p(?x,?z);
}
```

Concerning querying OWL ontologies, Bossam supports queries in the Buchingae rule language. It is actually a rule without a conclusion-part, just like the defquery construct of OWLJessKB. For example, in order to retrieve all the instances of the Region class that have the region1 instance in their subRegionOf property, we should write

```
query region-instances is Region(?x) and subRegionOf(?x, region1);
```

OWLIM

OWLIM is a high-performance Storage and Inference Layer (SAIL) for Sesame (Broekstra, Kampman, & van Harmelen, 2002), which performs reasoning based on forward-chaining of entailment rules. The inferencing procedure is based on the TRREE dedicated OWL reasoning engine. OWLIM is available in two versions:

- SwiftOWLIM performs reasoning and query evaluation in memory, while a reliable persistence strategy assures data preservation, consistency, and integrity.
- BigOWLIM operates directly with binary persistence files, which allows it to scale to billions of statements.

The TRREE engine can be configured with a set of inference rules, which determines the supported semantics. Each rule has a set of premises, which conjunctively define the body of the rule. The premises are RDF statements, which can contain free variables. The rule head contains one or more consequences, each of which is an RDF statement, without free variables (safe rules). The implementation of TRREE relies on a compile stage, when the entailment rules are compiled into chunks of Java code that are merged together to generate the main entry point for the reasoner. To exemplify on the syntax of the rule language of OWLIM, we present the syntax of the transitive entailment rule.

```
Id: owl_TransProp
    p  <rdf:type>  <owl:TransitiveProperty>
    x  p  y                          [Constraint x != y]
    y  p  z                          [Constraint y != z]
   -------------------------------
    x  p  z
```

Queries are encoded in the SeRQL (SeRQL, 2008) language. To give an example, the query for retrieving all the instances of the Region concept that have the region1 instance in their subRegionOf property is defined as

```
select x
        from {x} rdf:type {Region},
        {x} subRegionOf {region1}
```

BaseVISor

BaseVISor is a forward-chaining inference engine based on a Rete network optimized for the processing of RDF triples that is able to process RuleML (RuleML, 2008) rules containing n-ary predicates.

The Rete-based forward chaining inference engine is similar to Jess and CLIPS. The primary difference is that it uses a simple data structure for its facts rather than arbitrary list structures, which permits greatly enhanced efficiency in pattern matching which is at the core of a Rete network. The facts are defined as triples with subjects, predicates and objects, following an XML syntax. For example, the t1 triple of the region ontology is transformed as

```
<triple>
        <subject resource=”Region”/>
        <predicate resource=”rdf:type”/>
        <object resource=”owl:Class”/>
</triple>
```

Rules are defined in the RuleML syntax within a rulebase with each rule consisting of a body and a head element. BaseVISor has the built-in ability of converting such RuleML rulebases into native BaseVISor code. In that way, the transitive entailment is represented as

```
<rule name="rdfp4">
        <body>
                <triple>
                        <subject variable="p"/>
                        <predicate resource="rdf:type"/>
                        <object resource="owl:TransitiveProperty"/>
                </triple>
                <triple>
                        <subject variable="u"/>
                        <predicate variable="p"/>
                        <object variable="v"/>
                </triple>
                <triple>
                        <subject variable="v"/>
                        <predicate variable="p"/>
                        <object variable="w"/>
                </triple>
        </body>
        <head>
                <debug><param>rdfp4</param></debug>
                <assert>
                        <triple>
                                <subject variable="u"/>
                                <predicate variable="p"/>
                                <object variable="w"/>
                        </triple>
                </assert>
         </head>
</rule>
```

It is possible to query the facts using the query element and placing in its content one or more triples containing variables. The result of a query is a list of variable bindings that satisfy the constraints of the query. For example, the query for retrieving the Region instances that have the region1 in their subRegionOf property can be defined as

```
<query name="region-instances">
        <triple>
                <subject variable="X"/>
                <predicate resource="rdf:type"/>
                <object resource="Region "/>
        </triple>
        <triple>
                <subject variable="X"/>
                <predicate resource="subRegionOf"/>
                <object resource="region1"/>
        </triple>
</query>
```

DL VS RULE-BASED OWL REASONING

There is a lot of debate about the suitability of the LP paradigm in the domain of Semantic Web. In this section we are focused on the comparison of the DL reasoning paradigm that embodies the Classical Logic, and the rule-based OWL reasoning paradigm that embodies the Datalog paradigm.

Open and Closed-World Semantics

One of the most controversial topics of discussion is related to the open nature of the Semantic Web. Since in this environment it is difficult to have the complete information modeled into an application, the assumption that the lack of information is equivalent to negative information (negation as failure) seems improper. To this end, the Classical paradigm seems more appropriate for modeling in the Semantic Web since it follows the open-world assumption (OWA), that is, unstated information does not necessarily mean negated information. On the other hand, the Datalog paradigm follows the closed-world assumption (CWA), that is, all the relevant information is explicitly known and thus unprovable facts should be assumed not to hold. For example, consider the two statements friendOf(George, Nick) and friendOf(George, Peter) that denote the friends of George. The Datalog paradigm assumes that George has only these two friends, whereas the Classical paradigm does not exclude the possibility of George to have more friends.

Although the Classical paradigm seems more appropriate for the open nature of the Semantic Web, there are many cases that require closed-world reasoning, such as database applications or applications that require queries about negative information. In these cases the Datalog paradigm has an advantage since it natively supports such kind of reasoning. For example, consider a database with the students of a university department. In order to answer a query about whether a person is a student at the department, a form of closed-world reasoning is required, whereas such queries cannot be answered following the Classical paradigm of OWL. The notion of closed-world reasoning can be emulated in the Classical paradigm, based on the way the results are interpreted. In that way we can treat the query results of the Classical

paradigm as information that is "known to hold". For example, a query for the friends of George will return Nick and Peter and thus, we can say that these are the two persons that "is known to be George's friends". However the native closed-world reasoning ability of the rule paradigm is more intuitive and practical in the domains that require such type of reasoning.

Unique Name Assumption

In the Classical Logic two different name identifiers do not necessarily mean that they are different objects, in contrast to the Datalog paradigm where different identifiers always denote different objects (unique name assumption - UNA). The approach of Classical Logic fits better in the domain of Semantic Web where, due to the decentralized nature, people might use different identifies to denote the same individual. For example, OWL supports the notion of the functional property (owl:FunctionalProperty) that denotes that there cannot be two distinct values *y* and *z* such that the pairs (*x*,*y*) and (*x*,*z*) are both instances of the property. Thus, if there is a functional property hasBiologicalMother and two statements

hasBiologicalMother(George, Maria)
hasBiologicalMother(George, Eva),

then Maria and Eva will be inferred to be the same individuals. Such possibility does not exist in the Datalog paradigm that follows the UNA. However, there are critical situations where we want the UNA to hold, otherwise the system could result in false inferences due to mistakes either at the ontology level or at the instance level, such as names of seats in an airplane (de Bruijn et al., 2005). Although in Classical Logic the UNA does not hold by default, there is the possibility of expressing the UNA for a set of individual, for example the owl:AllDifferent OWL construct, whereas in Datalog different names always identify different objects. However, the native UNA behavior of rule engines seems more practical especially in cases with large number of instances.

Reasoning with Incomplete Information

The open nature of the Semantic Web requires also the ability to reason with incomplete information, that is, the ability to state something about a resource without providing the complete information. For example, it is useful to say that George has at least two friends, without stating who they are or the exact number of his friends. This case can be defined in OWL stating that George $\in \geq 2$ friendOf (cardinality restrictions). However, in Datalog such relationships can not be stated since the relationships can be expressed only over named individuals that exist in the KB (CWA). Even if two arbitrary individuals are generated to play the role of the two friends of George, this would incorrectly affect the total number of his friends due to the UNA.

The same holds in the case of a concept definition where local property range constraints are provided. For example, in the Classical paradigm we can state that the parents of a person are also instances of the class Person as Person $\sqsubseteq \forall$parent.Person (universal quantifiers). In Datalog such constraint can be represented as

$\leftarrow$ Person(x) $\wedge$ parent(x, y) $\wedge \neg$Person(y).

However, the two modeling paradigms have a conceptually different intention. In Classical Logic, the notion of constraint allows performing additional inferences. Thus, the statement parent(George, Peter) is valid in the Classical paradigm, inferring also that Person(Peter), whereas in Datalog is not valid. In Datalog, the notion of constraints is more similar to the notion of the database constraints in order to restrict the value into a specific type. Furthermore, the above restriction can be modeled in Datalog as

$$\text{Person}(y) \leftarrow \text{Person}(x) \wedge \text{parent}(x, y).$$

Such a rule requires that every person has a known parent, and thus can only be applied on named individuals that exist in the KB. Therefore, OWL offers more inferencing capabilities, while Datalog has native support for capturing application inconsistencies.

Reasoning Complexity

The size of the web is overwhelming and one of the most important requirements in real world ontology applications is the ability of reasoning over large number of instances (ABox reasoning). The Tableaux-based algorithms, on which the DL reasoning paradigm is based, provide efficient and optimized TBox reasoning, that is the computation of the subsumption hierarchy, but they do not provide efficient ABox reasoning and consequently, query answering capabilities. For example, the reasoning in OWL DL is NEXPTIME-complete (Horrocks & Patel-Schneider, 2003). Although there are many efforts towards the development of efficient reasoning algorithms (Haarslev & Moller, 1999; Horrocks et al., 2004a; Hitzler & Vrandecic, 2005), efficient ABox reasoning is still an open issue.

In contrast to the increased complexity of the Classical Logic, the polynomial-time reasoning complexity of the Datalog paradigm is very attractive. However, a direct comparison of the reasoning complexity of the two paradigms seems unfair. The DL reasoning paradigm has increased inferencing capabilities, enabling it to reason over the incomplete information of the open environment of the Semantic Web. On the other hand, the polynomial reasoning time of Datalog is achieved when the query answering is performed against a fixed program, that is, over unchanged instances. Otherwise, the complexity becomes exponential. The same holds for more expressive Datalog versions, such as the Datalog$^{\neg\vee}$ with negation as failure in the body and disjunction in the head of rules, where complexity goes beyond polynomial time (Eiter, Gottlob & Mannila, 1997). Therefore, the reasoning complexity is proportional to the expressivity we want to incorporate during modeling.

CONCLUSIONS

The advantages that stem from the rule-based representation paradigm, such as the flexible and declarative way of specifying knowledge or the simplicity it offers during transfer of knowledge between different parties, have been quickly identified and rules have become an inextricable part of many real-world applications. The importance of rules has motivated many standardization organizations to work on the recommendation of a common rule standard, such as the RuleML (RuleML, 2008), W3C (W3C, 2008) and the OASIS (OASIS, 2008) initiatives. It is worth to mention the OMG Production Rule Representation (OMG, 2008) that works towards the specification of a standard for platform-independent expression of production rules.

In this chapter we addressed the basic principles behind the utilization of rules in order to handle OWL ontological information. Such utilization is based on two approaches. Firstly, rules can be considered as alternatives to the OWL language, mapping a subset of DL into the LP paradigm. Secondly, rules can be combined with the OWL language, enhancing the semantics. Such combinations can be either hybrid, based on the inferencing capabilities of a DL component and a rule engine to run the rules, or homogeneous, where rule and ontology predicates are treated homogeneously as a single logic language with new semantics.

The chapter is mainly focused on presenting the practical aspects of the utilization of rule engines towards the development of entailment-based OWL reasoning (EBOR) systems for handling the OWL language. In fact, this is the first step towards the development of a homogeneous system for building rules on top of ontologies, as well as ontologies on top of rules. Towards this end, we presented the notion of RDF and OWL entailment rules and a classification of the EBOR paradigm into extended and native, according to the nature of the underlying rule engine. We presented three extended (OWLJessKB, F-OWL and O-DEVICE) and four native (Jena2, Bossam, OWLIM and BaseVISor) EBOR systems, describing (a) the distinctive implementation aspects of each system, (b) their rule languages, presenting the way the RDF/OWL entailments are implemented, and (c) the query languages they use in order to retrieve ontological resources. F-OWL and O-DEVICE are motivated by the similarities that exist between OWL and the object-oriented paradigm, whereas the other approaches follow a triple-based functionality. Furthermore, F-OWL uses a Prolog-like and Jena2 can use a backward chaining inference engine, whereas the other approaches follow the production rule paradigm where the inferred triples are computed and stored a priori in the KB.

We presented also a comparison regarding the DL and the rule-based reasoning paradigms. Since OWL is based on the DL formalism, existing sound and complete DL reasoning systems can be used for reasoning on ontologies. However, they have some limitations, such as inefficient ABox reasoning capabilities or the inability to handle complex and large rule programs, since they are not rule engines. Thus, although the rule-based reasoning paradigm offers less expressivity, it is closer to real-world application domains, such as database applications where there is the need for type constraints rather than high inferencing capabilities. Nevertheless, each modeling paradigm has its own weaknesses and strengths, depending on the application domain it is applied.

REFERENCES

Antoniou, G., & Harmelen, F. (2004). A Semantic Web Primer. Cooperative Information Systems. MIT Press.

Antoniou, G., Damasio, C.V., Grosof, B., Horrocks, I., Kifer, M., Maluszynski, J., & Patel-Schneider, P.F. (2005). Combining Rules and Ontologies. A Survey. Reasoning on the Web with Rules and Semantics, REWERSE Deliverables.

Baader, F. (2003). The Description Logic Handbook : Theory, Implementation and Applications. Cambridge University Press.

Baader, F. and Sattler, U. (2001). An overview of tableau algorithms for description logics. Studia Logica, 69(1), 5-40. Springer.

Baral, C. (2003). Knowledge Representation, Reasoning, and Declarative Problem Solving. Cambridge. Cambridge University Press.

Beckett, D., & McBride, B. (2004). RDF/XML Syntax Specification (Revised). Retrieved April 14, 2008, from http://www.w3.org/TR/rdf-syntax-grammar/.

Berners-Lee, T., Hendler, J., & Lassila, O. (2001). The Semantic Web. Scientific American Magazine, 284(5), 34-43.

Broekstra, J., Kampman, A., and van Harmelen, F. (2002). Sesame: A generic architecture for storing and querying rdf and rdf schema. International Semantic Web Conference (pp. 64-68).

de Bruijn, J., Eiter, T., & Tompits, H. (2008). Embedding approaches to combining rules and ontologies into autoepistemic logic. 11th International Conference on Principles of Knowledge Representation and Reasoning (KR2008), Sydney, Australia, September 16-19.

de Bruijn, J., Lausen, H., Polleres, A., & Fensel, D. (2006). The web service modeling language: An overview, Proc. 3rd European Semantic Web Conf., June 2006.

de Bruijn, J., Lara, R., Polleres, A., & Fensel, D. (2005). OWL DL vs. OWL Flight: Conceptual Modeling and Reasoning for the Semantic Web. In Proceedings of the International Conference on World Wide Web (pp. 623-632), ACM Press.

Donini, F. M., Lenzerini, M., Nardi, D., & Schaerf, A. (1998). AL-log: Integrating Datalog and Description Logics. Intelligent and Cooperative Information Systems, 10, 227-252.

Drabent, W., Henriksson, J., & Maluszynski, J. (2007). HD-rules: A Hybrid System Interfacing Prolog with DL-reasoners. In Proceedings of Applications of Logic Programming to the Web, Semantic Web and Semantic Web Services. Vol. 287 (pp. 76-90). CEUR-WS.

Eiter, T., Gottlob, G., & Mannila, H. (1997). Disjunctive Datalog. ACM Transactions on Database Systems, 22(3), 364-418.

Eiter, T., Lukasiewicz, T., Schindlauer, R., & Tompits, H. (2004). Combining Answer Set Programming with Description Logics for the Semantic Web. In Proceedings of the International Conference of Knowledge Representation and Reasoning (pp. 141-151). Morgan Kaufmann.

Forgy, C.L. (1982). Rete: A Fast Algorithm for the Many Pattern/Many Object Pattern Match Problem. Artificial Intelligence, 19(1), 17-37.

Friedman-Hill, E. (2003). Jess in Action. Manning, Greenwich.

Grant, J., & Beckett, D. (2004). RDF Test Cases. Retrieved April 14, 2008, from http://www.w3.org/TR/rdf-testcases/.

Grosof, B. N., Horrocks, I., Volz, R., & Decker, S. (2003). Description Logic Programs: Combining Logic Programs with Description Logic. In Proceedings of the International Conference on World Wide Web (pp. 48-57). ACM Press.

Haarslev, V., & Moller, R. (1999). An Empirical Evaluation of Optimization Strategies for ABox Reasoning in Expressive Description Logics. In Proceedings of the International Workshop on Description Logics. Vol. 22. CEUR-WS.

Haarslev, V., & Moller, R. (2003). Racer: A Core Inference Engine for the Semantic Web. In Proceedings of the International Workshop on Evaluation of Ontology-based Tools (pp. 27-36).

Hayes, P. (2004). RDF Semantics. Retrieved April 14, 2008, from http://www.w3.org/TR/rdf-mt/.

Heymans, S., Predoiu, L., Feier, C., de Bruijn, J., & Van Nieuwenborgh, D. (2006). G-Hybrid Knowledge Bases. In Proceedings of the Workshop on Applications of Logic Programming in the Semantic Web and Semantic Web Services. Vol. 196. CEUR-WS.

Hitzler, P., & Vrandecic, D. (2005). The SCREECH OWL Reasoner - Scalable approximate ABox reasoning for OWL. International Semantic Web Conference, Software Demo.

Horrocks, I., & Patel-Schneider, P. F. (2003). Reducing OWL Entailment to Description Logic Satisfiability. In Proceedings of the International Semantic Web Conference (pp. 17-29). Springer.

Horrocks, I., Li, L., Turi, D., & Bechhofer, S. (2004a). The Instance Store: Description Logic Reasoning with Large Numbers of Individuals. In Proceedings of the Workshop on Description Logics. Vol. 104 (pp. 31-40). CEUR-WS.

Horrocks, I., Patel-Schneider, P. F., Boley, H., Tabet, S., Grosof, B., & Dean, M. (2004b). SWRL: A Semantic Web Rule Language Combining OWL and RuleML. Technical report, W3C

Member Submission.

Horrocks, I., & Sattler, U. (2004c). Decidability of SHIQ with Complex Role Inclusion Axioms. Artificial Intelligence, 160(1), 79-104. Elsevier.

Horrocks, I., Kutz, O., & Sattler, U. (2006). The Even More Irresistible SROIQ. In Proceedings of the International Conference on Principles of Knowledge Representation and Reasoning (pp. 57-67). AAAI Press.

Horst, H.J. (2004). Extending the RDFS Entailment Lemma, In Proceedings of the International Semantic Web Conference (pp. 77-91). Springer.

Horst, H.J. (2005). Completeness, Decidability and Complexity of Entailment for RDF Schema and a Semantic Extension Involving the OWL Vocabulary, Journal of Web Semantics, 3(2–3), 79-115.

Hustadt, U., Motik, B., & Sattler, U. (2004). Reducing SHIQ-Description Logic to Disjunctive Datalog Programs. In Proceedings of the International Conference on Knowledge Representation and Reasoning (pp. 152-162). AAAI Press.

Kattenstroth, H., May, W., & Schenk, F. (2007). Combining OWL with F-Logic Rules and Defaults. In Proceedings of the International Workshop on Applications of Logic Programming to the Web, Semantic Web and Semantic Web Services. Vol. 287 (pp. 60-75). CEUR-WS.

Kifer, M., Lausen, D., & Wu, J. (1995). Logical Foundations of Object-Oriented and Frame-Based Languages. Journal of ACM, 42(4), 741-843.

Kiryakov, A., Ognyanov, D., & Manov, F. (2005). OWLIM - A Pragmatic Semantic Repository for OWL. In Proceedings of the International Workshop on Scalable Semantic Web Knowledge Base Systems (pp. 182-192). Springer.

Kopena, J.B., & Regli, W.C. (2003). DAMLJessKB: A Tool for Reasoning with the Semantic Web. 2nd Intl. Semantic Web Conference (ISWC2003).

Levy, A.Y., & Rousset, M. (1998). Combining Horn Rules and Description Logics in CARIN. Artificial Intelligence, 104(1-2), 165-209. Elsevier.

Matheus, C., Baclawski, K., & Kokar, M. (2006). BaseVISor: A Triples-Based Inference Engine Outfitted to Process RuleML and R-Entailment Rules. In Proceedings of the International Conference on Rules and Rule Languages for the Semantic Web.

McBride, B. (2001). Jena, Implementing the RDF Model and Syntax Specification, In Proceedings of the International Workshop on the Semantic Web. Vol. 40. CEUR-WS.

McGuinness, D. L. & Harmelen, F. (2004). OWL Web Ontology Language Overview, W3C Recommendation, Retrieved April 14, 2008, from http://www.w3.org/TR/owl-features/.

Meditskos, G., & Bassiliades, N. (2007). A Semantic Web Service Discovery and Composition Prototype Framework Using Production Rules. Workshop on OWL-S: Experiences and Future Developments, European Semantic Web Conference, Innsbruck.

Meditskos, G., & Bassiliades, N. (2008a). Combining a DL Reasoner and a Rule Engine for Improving Entailment-based OWL Reasoning. International Semantic Web Conference (ISWC). Karlsruhe, Germany.

Meditskos, G., & Bassiliades, N. (2008b). A Rule-Based Object-Oriented OWL Reasoner. IEEE Transactions on Knowledge and Data Engineering, 20, 397-410.

Mei, J., Lin, & Z., Boley, H. (2007). ALC: An Integration of Description Logic and General Rules, In Proceedings of the Web Reasoning and Rule Systems (pp. 163-177). Springer.

Minsu, J., & Sohn, J.C. (2004). Bossam: An Extended Rule Engine for OWL Inferencing. In Proceedings of Rules and Rule Markup Languages for the Semantic Web (pp. 128-138). Springer.

Motik, B., Grau, B.C., Horrocks, I., Wu, Z., Fokoue, A., & Lutz, C. (2008). OWL 2 Web Ontology Language Profiles. OWL Working Group.

Motik, B., Horrocks, I., & Sattler, U. (2007). Adding Integrity Constraints to OWL, Proc. of the Third OWL Experiences and Directions Workshop. CEUR.

Motik, B., Horrocks, I., Rosati, R., & Sattler, U. (2006). Can OWL and Logic Programming Live Together Happily Ever After?. In Proceedings of the International Semantic Web Conference (pp. 501-514). Springer.

OASIS (2008). Organization for the Advancement of Structured Information Standards. Retrieved April 14, 2008, from www.oasis-open.org/.

OMG (2008). The Object Management Group (OMG), Retrieved April 14, 2008, from http://www.omg.org/

OWL-S (2004). Semantic Markup for Web Services, Retrieved April 14, 2008, from http://www.w3.org/Submission/OWL-S/

Paolucci, M., & Sycara, K. (2003). Autonomous Semantic Web Services. IEEE Internet Computing, 7(5), 34-41.

Patel-Schneider, P. F., & Horrocks, I. (2006). Position Paper: A Comparison of Two Modelling Paradigms in the Semantic Web. In Proceedings of the International Conference on World Wide Web (pp 3-12). ACM Press.

Prud'hommeaux, E., & Seaborne, A. (2008). SPARQL Query Language for RDF. W3C

Recommendation. Retrieved April 14, 2008, from http://www.w3.org/TR/rdf-sparql-query/.

Riley, G. (1991). CLIPS: An Expert System Building Tool. Proceedings of the Technology 2001Conference, San Jose, CA.

Roman, D., Keller, U., Lausen, H., de Bruijn, J., Lara, R., Stollberg, M., Polleres, A., Feier, C., Bussler, C., & Fensel, D. (2005). Web Service Modeling Ontology, Applied Ontology, 1(1), 77-106.

Rosati, R. (1999). Towards expressive KR systems integrating datalog and description logics: preliminary report. In Proceedings of the International Workshop on Description Logics. Vol. 22 (pp. 160-164). CEUR-WS.

Rosati, R. (2005). Semantic and Computational Advantages of the Safe Integration of Ontologies and Rules. In Proceedings of Principles and Practice of Semantic Web Reasoning (pp. 50-64). Springer.

Rosati, R. (2006). DL+log: Tight Integration of Description Logics and Disjunctive Datalog. In Proceedings of the International Conference on Principles of Knowledge Representation and Reasoning (pp 68-78). AAAI Press.

RuleML (2008). The Rule Markup Initiative. Retrieved April 14, 2008, from http://www.ruleml.org/.

Sagonas, K., Swift, T., & Warren, D.S. (1994). XSB as an Efficient Deductive Database Engine, ACM SIGMOD Record, 23(2), 442-453.

SeRQL (2008). The SeRQL query language. User Manual. Retrieved April 14, 2008, from http://www.openrdf.org/doc/sesame/users/ch06.html

Sirin, E., Parsia, B., Grau, B. C., Kalyanpur, A., & Katz, Y. (2007). Pellet: A Practical OWL-DL Reasoner. Web Semantics: Science, Services and Agents on the World Wide Web, 5(2), 51-53.

Swift, T. (2004). Deduction in Ontologies via ASP. In Proceedings of Logic Programming and Nonmonotonic Reasoning (pp. 275-288). Springer.

Tsarkov, D. and Horrocks, I. (2006). Fact++ description logic reasoner: System description. In Proceedings of Automated Reasoning (pp. 292-297). Springer.

Van Belleghem, K., Denecker, M., & De Schreye, D. (1997). A strong correspondence between description logics and open logic programming. In Proceedings of the International Conference on Logic Programming (pp. 346-360). MIT Press.

W3C (2008). The Semantic Web Activity. Retrieved April 14, 2008, from http://www.w3.org/2001/sw/.

Wagner, G., Antoniou, G., Tabet, S., & Boley, H. (2004). The Abstract Syntax of RuleML - Towards a General Web Rule Language Framework. In Proceedings of the International Conference on Web Intelligence (pp. 628-631). IEEE Computer Society.

Wang, K., Billington, D., Blee, J., & Antoniou, G. (2004). Combining Description Logic and Defeasible Logic for the Semantic Web, In Proceedings of Rules and Rule Markup Languages for the Semantic Web (pp. 170-181). Springer.

Yang, D., Kifer, M., & Zhao, C. (2003). FLORA-2: A Rule-Based Knowledge Representation and Inference Infrastructure for the Semantic Web. In Proceedings of the International Conference on Ontologies, Databases and Applications of Semantics (pp. 671-688). Springer.

Zou, Y., Finin, T., & Chen, H. (2004). F-OWL: An Inference Engine for Semantic Web. In Proceedings of the International Workshop on Formal Approaches to Agent-Based Systems (pp. 238-248). Springer.

KEY TERMS AND THEIR DEFINITION

1. Rule Engine: A computer program able to derive answers from a knowledge base based on a set of rules.
2. Semantic Web: The extension of the current Web where information is given well-defined meaning, enabling computers and people to work in better cooperation.
3. Ontology: A specification of a shared conceptualization using a formal language.
4. Web Ontology Language (OWL): The W3C recommendation for creating and sharing ontologies on the Web.
5. Rule-based OWL Reasoning: The process of reasoning about OWL ontologies based on a rule engine.
6. Description Logic Reasoning: The process of reasoning about OWL ontologies based on Description Logic algorithms (e.g. tableaux-based algorithms).
7. Entailment Rule: An inference rule that defines the information that should be derived based on existing ontological knowledge.

LIST OF ACRONYMS

CWA: Closed-World Assumption
DL: Description Logic
DLP: Description Logic Programs
EBOR: Entailment-based OWL Reasoning
E-EBOR: Extended Entailment-based OWL Reasoning
LHS: Left-Hand Side
LP: Logic Programming
N-EBOR: Native Entailment-based OWL Reasoning
OID: Object ID
OWA: Open-World Assumption
RHS: Right-Hand Side
SWRL: Semantic Web Rule Language
UNA: Unique Name Assumption