FIRE – A DESCRIPTION LOGIC BASED RULE ENGINE FOR OWL ONTOLOGIES WITH SWRL-LIKE RULES

KRUTHI BHOOPALAM

A THESIS
IN
THE DEPARTMENT
OF
COMPUTER SCIENCE AND SOFTWARE ENGINEERING

PRESENTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF MASTER OF COMPUTER SCIENCE
CONCORDIA UNIVERSITY
MONTRÉAL, QUÉBEC, CANADA

SEPTEMBER 2005

Your file Votre référence
ISBN: 0-494-10280-2
Our file Notre référence
ISBN: 0-494-10280-2

Abstract

Fire – A Description Logic Based Rule Engine for OWL Ontologies with SWRL-like Rules

Kruthi Bhoopalam

The present decade has seen significant progress towards realizing the vision of the Semantic Web. This progress has most often been seen in the levels of maturity reached by each *layer* in the architectural layers representing the Semantic Web vision. The *ontology layer* reached a substantial level of maturity with the OWL Web Ontology Language (OWL) being recommended by the World Wide Web Consortium (W3C) as the standard for representing ontologies on the Web. This move has triggered several other standardizations and led to interesting research results that have further strengthened the ontology layer. This has motivated the Semantic Web community to venture further towards the *rules layer* in the vision. One of the interesting lines of research in this context is to extend OWL with rules both syntactically and semantically and providing a sound, complete and terminating reasoning support for the extension. Our present work corresponds to this line of research.

We investigate the problem of providing a description logic (DL) based rule reasoning support for OWL ontologies extended with rules. Guided by the Semantic Web Rule Language (SWRL) extension to OWL, we recognize a rule language called the SWRL-like rule language as a rule-extension to OWL. The SWRL-like rule language has a similar syntax as SWRL but differs in its semantics. We propose the system **Fire**, a rule reasoning engine for the extension of OWL ontologies with SWRL-like rules. The reasoning procedure proposed for the Fire system follows *active domain semantics* to ensure termination. The reasoning procedure is conjectured to be sound and complete based on the approach followed in the *CARIN* system. Contrary to several existing translation-based approaches for reasoning with OWL ontologies combined with rules, our proposal provides direct *DL based* rule inferencing that is synchronous with the OWL inferencing. This synchronous integration with a DL reasoner offers immediate feedback about rule consequences in the OWL knowledge base (KB). The proposal is supported with a prototype Java implementation

of the Fire system. The prototype implements the *RETE* algorithm for *pattern matching*. Our experiments with the pattern matcher algorithm indicate higher efficiency and speed of the implemented RETE algorithm in matching DL facts with SWRL-like rule patterns, compared to a naïve approach to pattern-matching. The implemented prototype is sound based on the sound and complete reasoning of the DL reasoner RACER. Termination is ensured by the active domain semantics. The prototype does not guarantee completeness of reasoning due to the unavailability of an important OWL reasoning service from any of the existing DL reasoners.

Acknowledgments

***Maathru* devo bhava.** (Respect thy *mother* like God)
***Pithru* devo bhava.** (Respect thy *father* like God)
***Aachaarya* devo bhava.** (Respect thy *teacher* like God)

I dedicate this work to my parents and my supervisor.

My profound thanks to my supervisor Dr.Volker Haarslev for his guidance, help and constant encouragement. In the past two years, his direction and support at every step of my work have helped me a lot in my learning.

My mother Sumitra, my father Krishnamurthy, and my younger brother Rahul have always been a tremendous source of motivation, strength, and encouragement for my education. My gratitude to my family is immense.

My heartfelt thanks to my colleagues and friends for their support and encouragement academically or otherwise. In particular, Xi Deng - for assisting me with all my doubts with a smile, Hemanth - for always being there and encouraging me, Sabeel - for answering my every small question without getting bored, Anup - for teaching me a lot, Israat - for all the discussions and help, Ram - for the ever-cheerful support, Tejas - for being so good, Gurudath - for being ever-entertaining despite his mood, Uma - for being so caring and Vasavi - for all the care and support. A special thanks to my close friends for those never-ending jokes which always refreshed my mind. Montreal has been home because of them.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

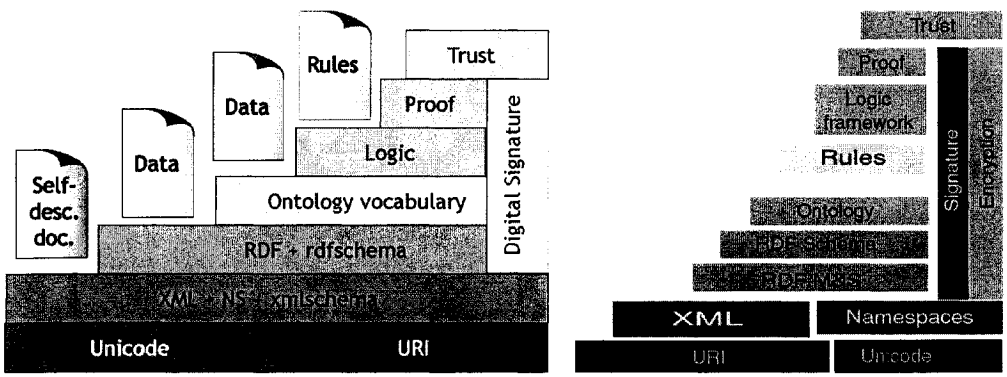
The Web of today was designed with a goal that the vast information it holds will not only be used for human-to-human communication but also for enabling machines to participate and help. This goal is reflected by the design issues of the Web[1] in the discussion, 'A roadmap to the Semantic Web'. But the content on the Web has grown in a way that most of it is designed mainly for human consumption. The ways in which the data is structured are also for human consumption, which are not evident to machine processing of the data. Tim Berners-Lee, the creator of the World Wide Web, envisioned the Semantic Web project to overcome this obstacle for realizing the full power of the Web. The goal of this project is to extend the Web in a way to utilize the full participation and help of machines. The approach is to (i) develop languages to represent and structure data, including their meaning (semantics) and (ii) to establish a universal medium for exchange of information. This should be done in a formal way that is suited for machine consumption (Such representations of information are in addition to the existing versions of representations designed for human consumption). Machines could then be engineered to better *understand* such information enhanced with *formal semantics*, to perform automated reasoning with the Web content, and carry out more intelligent tasks on behalf of the user. At this point it should be clear that the concept of machine-understandable information does not indicate the ability of the machine to comprehend human language. It only indicates a machine's ability to solve a well-defined problem by performing well-defined operations on existing well-defined data.[2] It involves humans in the process of *defining* the data and the operations over them, in a formal way that gives the machine its ability to *understand* the *meaning* of

[1] http://www.w3.org/DesignIssues/Overview.html
[2] http://www.w3.org/DesignIssues/Semantic.html

the information it processes.

1.1 Progress Towards the Semantic Web Vision

Figure 1(a) pictorially depicts the Semantic Web vision and is popularly known as the *Semantic Web layer cake* first used by Tim Berners-Lee. It shows the architectural layers of the vision. Each layer corresponds to a level of knowledge representation and structuring that is needed to realize the vision, with the standard technologies that are enabling them. There have been growing efforts in realizing this vision. Owing to these efforts and to the recent broad consensus in the community to include *rules* along with ontologies in the Semantic Web architecture, a more refined layer diagram resulted [ADG+05]. The *refined* layer diagram is shown in Figure 1(b), which is borrowed from Tim Berners-Lee's keynote talk at ISWC2003[3]. We present below, a brief description of these layers in the architecture diagram represented by both Figures 1(a) and 1(b).



(a) (b)

Figure 1: The architectural layers in the Semantic Web vision.

- *The XML (eXtensible Markup Language) layer:* XML allows to mark-up and structure arbitrary content by means of nested, attributed elements. The structuring has no particular semantics to indicate what the structure means. XML plays the role of just a syntax carrier and this layer corresponds to a basic *syntax layer.*

[3]http://www.w3.org/2003/Talks/1023-iswc-tbl/Overview-1.html

- *The RDF (Resource Definition Framework) Model and Syntax layer:* This layer corresponds to the meaning of data. RDF allows encoding, exchange, and reuse of structured metadata. In principle, information is represented by generic means, say by directed partially labelled graphs that may be serialized using XML. Contrary to XML, RDF allows assigning global identifiers to such information resources and allows one resource document to refer to and extend statements made in other resource documents. This feature mainly motivates for its use in this *data layer.*

- *The Ontology layer:* This layer corresponds to the formal common agreement about the meaning of data in terms of ontologies. Ontologies formally describe the shared conceptualizations of particular domains of interest. This description can be used to describe structurally heterogeneous and distributed information sources found on the Web. Ontologies help both people and machines to communicate concisely by defining shared and common domain theories and vocabularies. They support the exchange of semantics as well as syntax.

- *The Rules layer and the Logic layer:* These two layers together allow for querying meaningful data. Rules in these layers enable automated reasoning with the formally specified, structured data, to enable interesting inferences.

- *The Proof layer:* This layer has been conceived to allow the explanation of given answers generated by automated agents. This will require the translation of agents' internal reasoning mechanisms into some unifying proof representation language. It supports the exchange of proofs between such agents, enabling a common understanding of how a desired information is derived.

In recent years, the Semantic Web research community has seen significant advancements towards realizing this architectural vision. The advancements have been in the form of standardizations of languages and Web technologies that enable this vision. The progress is seen in terms of levels of maturity reached in each layer. The OWL Web Ontology Language (OWL)[4] which is a description logic[5] [BCM+03, BN03] based language, has been one such advancement which is now the World Wide Web consortium recommended standard for representing ontologies on the Web. OWL provides three sublanguages with

[4]http://www.w3.org/2004/OWL/

[5]Description logics are also popularly known as terminological logics or concept languages.

increasing expressivity, the OWL Lite sublanguage, the OWL DL sublanguage, and the OWL Full language. Details of these sublanguages are presented in Section 1.3.

1.2 From Ontology to DL Based Rules

The Semantic Web and the description logics (DL) communities have seen dedicated state-of-the-art reasoners like RACER[HM01], FaCT[Hor98], Pellet[6] among others, providing efficient *decidable* reasoning over portions of the OWL language (specifically OWL DL). With OWL, and the significant progress its standardization has triggered, the ontology layer has attained a substantial level of maturity towards the vision. Many complex relationships between concepts can be expressed in OWL. But, when it comes to expressing certain relationships between roles, the expressivity offered by OWL is not enough. Consider the example to express the definition of a role $hasUncle$ as the composition of roles *hasParent* and $hasBrother$. Another example (borrowed from [Rec02]) to express *propagation across transitive roles*: a fracture $locatedIn$ the femur bone, where femur bone $isPartOf$ the leg, means a fracture is $locatedIn$ the leg. Here the $isPartOf$ is transitive. These ideas cannot be expressed in OWL without additional expressive features. Non-availability of known DL algorithms for decidable reasoning with the full OWL language and a need for expressivity not offered even by the full OWL language, have motivated research interests towards extending OWL DL with more expressive features. The broad consensus in the community indicates that such an extension should involve adding *rules* over the ontology layer. A move towards this extension points us towards several existing proposals for integrating ontologies and rules.

Generally, a *rule* is composed of an *antecedent* and a *consequent*, both made up of zero or more *atoms*.[7] An atom is made up of a *predicate* with an arity of one or more. Atoms can refer to the elements of a knowledge base (KB) and specify conditions involving them. So, basically the body and head of rules specify certain *conditions* and rules are used to derive new inferences about the knowledge represented by a KB.

Rules are logically treated in two ways namely, (i) as *trigger rules* and (ii) as *logical implications*. Trigger rules are popular with rule based knowledge representation systems and expert systems. Trigger rules usually operate only in one direction i.e., from antecedent

[6] http://www.mindswap.org.2003/pellet/index.shtml

[7] Throughout the thesis, *body* is synonymously used with *antecedent*, and *head* is synonymously used with *consequent*.

to consequent. Their intended meaning is simple and can be read as: *if the conditions in the antecedent are known to hold, then conclude that the conditions in the consequent also hold.* Whereas, the logical implications operate in both directions. That is, the contraposition law[8] holds on such rules. Their intended meaning can be read as: *whenever the conditions specified in the antecedent hold, then the conditions specified in the consequent must also hold.* The contraposition law gives the additional meaning that can be read as: if the conditions in the consequent *do not* hold, then the conditions in the antecedent also *do not* hold. Both kind of rules mentioned above offer added expressivity to the DL based ontology language it extends, enabling it to express complex relationships between its elements.

Two recent papers [ADG+05] and [Ros05] provide classifications of existing proposals for integrating ontologies and rules. [ADG+05] surveys work on combining rules and ontologies for the Semantic Web. The classification suggested is based on the degree of integration of the ontologies and rules: (i) *hybrid approach:* it strictly separates rule predicates from the ontology predicates that can appear in rule antecedents, and offers reasoning by interfacing existing rule and ontology reasoners and (ii) *homogenous approach:* it embeds both ontology and rules in a common logical language and does not distinguish between rule and ontology predicates appearing in rule bodies and offers reasoning either by a general reasoner of the common logic language or by constructing a specialized reasoner for the rule language. [Ros05] mentions *hybrid systems* constituted of two or more subsystems, each dealing with a distinct portion of the KB and using specific representation formalisms and reasoning procedures. Following these lines we classify only those existing proposals that integrate *DL based* ontologies with rules.

- **Hybrid approach:** This approach combines a DL ontology language with an existing rule language (mostly Datalog or its extensions). The rule component and the ontology component of the hybrid KB are treated distinctly. The ontology remains unchanged, but rules are built upon them. The interaction between the two components is obtained by enforcing some constraints on the predicates (both rule and ontology) in rule bodies and on the individuals/constants that the variables in these predicates can be bound with. Generally, in systems following this approach the set

[8] According to the *law of contraposition* the logical implication $p \rightarrow q$ and its contrapositive $\neg q \rightarrow \neg p$ are logically equivalent, where p and q are logical sentences/statements.

of symbols used as concept and role names by the ontology (i.e., ontology predicates) have an alphabet that is *disjoint* with the set of predicate symbols used by rule atoms in the rule language (i.e., rule predicates). But they share a common alphabet for variables and individuals/constants. The reasoning support for the hybrid knowledge is by interfacing existing ontology reasoners with existing rule reasoners. This approach is well-suited for the construction of tools for accessing heterogenous information systems. It benefits existing systems that are structurally represented by a DL based language, having application-specific rule programs and having a need to integrate both.

- **Homogenous approach:** This approach basically works by treating the ontology and rules homogeneously in a single logic language. From a DL point of view, this can be further classified as follows:
  - **Translation based Approach.** The ontology and rules are both translated to be embedded in a common logical language preserving their individual semantics. Reasoning support is provided by a general reasoner of the common language.
  - **DL based Approach.** The rule language in systems following this approach is generally a strict syntactic and semantic extension of the DL ontology language. The extension is itself an ontology language and is allowed to express the knowledge of the KB. Both ontology and rules use the common alphabet for the set of predicate symbols, variables and constants. Reasoning support for the ontology is extended by constructing a specialized reasoner to handle rule inferencing.

Our present work in the thesis is relevant to the line of research introduced above, focussing on the rules layer from the ontology layer in the Semantic Web architecture in Figure 1(b). We follow the *homogenous DL based approach* described above to combine the DL based OWL language with a rule language. For the rule language, we are guided by the recent proposal, the Semantic Web Rule Language (SWRL) [HPB+04] which combines the OWL DL and OWL Lite sublanguages of the OWL with the Unary/Binary Datalog RuleML sublanguages of the Rule Markup Language[9]. The SWRL rules are treated as *logical implications* described earlier in this section. We recognize a rule language that is

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

similar to SWRL in syntax, but a little different in semantics, which we call as the **SWRL-like** rule language. The main difference is the treatment of rules as *trigger rules*. We concentrate our work on the extension of OWL with this SWRL-like rule language and propose a reasoning system, **Fire**, for the extension. We present a prototype that partially implements the proposed design of the Fire reasoning engine.

The following sections set the stage for understanding the rest of the thesis, by discussing the ontology language OWL and the rule language SWRL. We then identify the SWRL-like rule language on which this thesis focuses, describe why it is called so and how it differs from SWRL.

1.3 The OWL Web Ontology Language

The OWL Web Ontology Language [OWL04] is the W3C standard for representing ontologies on the Web. It is a stepping stone in the progress towards achieving the Semantic Web vision (see Figure 1). The OWL language is a DL based semantic markup language for publishing and sharing ontologies on the Web. It is developed as a vocabulary extension of RDF (the Resource Description Framework) and is derived from the DAML+OIL Web Ontology Language[10]. The syntax of the language enables defining and instantiating Web ontologies in the form of descriptions of *classes*, *properties* and their instances. The formal semantics of the language specifies how to derive the ontology's logical conclusions (facts not literally present in the ontology but *entailed* by the semantics).

OWL provides three sublanguages offering increasing levels of expressivity. Each is an extension of its simpler predecessor, both in what can be legally expressed and in what can be logically derived.

OWL Lite corresponds to the DL $\mathcal{SHIF}(D_n)$. It is a basic functional subset of the entire language allowing for classification hierarchies and simple constraint features.

OWL DL is that subset of OWL corresponding to the description logic $\mathcal{SHOIN}(D_n)$, for which decidable reasoning procedures exist. It is expressive enough to represent and model domains that have complex relationships between its identified classes. But the only relationship between properties that it allows to express is subsumption between atomic properties, i.e., being able to express something like, *hasFather* is a subPropertyOf *hasParent*.

[10] http://www.w3.org/TR/daml+oil-reference

Adding rules can increase the expressivity of OWL DL to express more complex relationship between properties, especially by using variables.
OWL Full has the maximum expressiveness, has the full syntax of OWL language and the syntactic freedom of RDF. There exists no known DL algorithms to decidably reason with OWL Full.

1.4 The Semantic Web Rule Language - SWRL

The SWRL [HPB+04] is a recent proposal to the W3C to extend OWL with rules. It is based on a combination of OWL DL and OWL Lite sublanguages of OWL with the Unary/Binary Datalog RuleML sublanguages of the Rule Markup Language[11]. The rule extension is by adding *rule axioms* to the set of OWL axioms. The proposal extends the OWL abstract syntax to include the syntax of these rules and the OWL model-theoretic semantics to provide a formal meaning for ontologies that include rules written in this syntax. The extension is strictly a syntactic and semantic extension, hence has a tight integration to OWL.

1.4.1 Syntax

A SWRL rule axiom is made up of an antecedent and a consequent, both consisting of zero or more atoms. A rule with either zero atoms in the body (unconditional facts that always hold), or zero atoms in the head (a trivially empty head that never holds) is not interesting for adding expressivity to OWL. Such rules are better expressed in OWL without rule constructs or as queries respectively. In a rule with multiple atoms in the body, the body is treated as a conjunction of its atoms. In a rule with multiple atoms in the head, the head also is treated as a conjunction. But such a rule can be easily transformed into multiple rules each with a single-atom head [HPB+04]. Rule atoms can be of the form:[12]

- $C(x)$ – where C is an OWL Class (a simple named class or a class description) or a data range[13] and x is either a variable, OWL individual or OWL data value.

[11] http://www.ruleml.org
[12] The abstract syntax of SWRL rule axioms, in a version of extended BNF notation, is given in http://www.w3.org/Submission/2004/SUBM-SWRL-20040521/#2
[13] http://www.w3.org/TR/owl-ref/#DataRange

- $P(x,y)$ – where P is an OWL Property (an object property or a datatype property), x is either a variable or an OWL individual and y is either a variable, an OWL individual or an OWL data value.
- $sameAs(x,y)$, $differentFrom(x,y)$ – where x, y are variables or OWL individuals.
- $builtIn(r,x,\ldots)$ – where r is a built-in relation and $x,\ldots$ are OWL data values.

$C(x)$ can be referred to as a *concept atom/class atom*, $P(x,y)$ can be referred to as a *role atom/property atom*. According to [HPB+04] variables in a rule are treated as universally quantified, limited to the scope of that rule. Rules must be *safe*, i.e., only variables that occur in the rule body may occur in the rule head[14]. The syntax of SWRL doesn't allow disjunction of atoms, negation of atoms or any non-monotonic features like negation as failure or defaults.

An informal human-readable syntax for these rules is also specified for ease of readability and a typical rule in this syntax would look like:

$$hasParent(?x,?y) \wedge hasChild(?y,?z) \Rightarrow hasSibling(?x,?z)$$

It states that an individual x having a parent y who in-turn has a child z, is a sibling of z.

1.4.2 Semantics

A SWRL rule is treated as a *logical implication* between its body and head. The intended meaning is that: *whenever the conditions in the rule body hold, then the conditions in the rule head must also hold.* It is important to recall that, the law of contraposition holds on this implication. Accordingly, if a rule holds, its contrapositive also holds (See Section 1.2). According to the SWRL proposal citeHPB04, the formal semantics for the rules are given by an extension of the OWL interpretation by defining *bindings*. These bindings map variables in rules to elements in the ontology domain. A rule is satisfied by an interpretation iff every binding that satisfies its body also satisfies its head. A rule body or rule head is satisfied if the conjunction of its atoms is satisfied. A concept atom $C(x)$ is said to be satisfied if x is an instance of the class description or data range C. A role atom $P(x,y)$ is

[14] *Unsafe* rules allow variables in the head that do not occur anywhere in the body. If this is allowed, the definition of bindings for variables cannot be defined correctly in this context. Also, termination of reasoning cannot be guaranteed as all the individuals in the KB could be used to bind the variable in the head. E.g., the rule $Child(?x) \Rightarrow isInnocent(?y)$ is unsafe.

said to be satisfied if x is related to y by role P. A $sameAs(x,y)$ atom is said to be satisfied if x is interpreted as the same object as y. A $differentFrom(x,y)$ atom is said to be satisfied if x and y are interpreted as different objects, and a $builtIn(r,x,\ldots)$ atom is said to be satisfied if the built-in relation r holds on the interpretations of the arguments. An interpretation satisfies an ontology iff it satisfies every axiom (OWL axioms and rule axioms) and fact in the ontology. It is important to again note that since a rule is treated as a *logical implication*, the implied meaning is that the *contraposition law* holds on the implication (rule). For example, whenever the rule:

$$hasParent(?x,?y) \wedge hasChild(?y,?z) \Rightarrow hasSibling(?x,?z)$$

holds, its contrapositive implication:

$$\neg hasSibling(?x,?z) \Rightarrow \neg(hasParent(?x,?y) \wedge hasChild(?y,?z))$$

also holds. It has been pointed out from the first version of the SWRL proposal that this kind of a rule-extension to OWL DL makes it undecidable [HPS04]. According to citeMSS04, the reason is that OWL DL is a logic with *tree model property* (i.e., any satisfiable KB in this logic has a model of a certain tree-shaped form). So, the satisfiability (i.e., existence of a model) of a KB in this logic can be decided by searching for only such tree-shaped models and ensuring termination of the search. Addition of SWRL kind of rules to this logic causes loss of this tree model property. This leads to undecidability of interesting reasoning problems for the combination of OWL DL and SWRL rules.

1.5 The SWRL-like Rule Language

The SWRL-like rule language that we consider here is very much like SWRL, sharing the SWRL syntax and the human-readable syntax that are discussed in Section 1.4. The difference is in the semantics. We describe the language and hightlight its differing semantics from SWRL.

1.5.1 Syntax

A SWRL-like rule is made up of an antecedent and a consequent, both consisting of one or more atoms.[15] The head of a rule consists of exactly one atom[16] and a rule body with multiple atoms is treated as a conjunction of its atoms.

Rule atoms can be of the form:

- $C(x)$ – where C is a simple *named* OWL Class and x is either a variable or an OWL individual.
- $P(x,y)$ – where P is an OWL object property and x, y are variables or OWL individuals.

Note that currently the syntax does not allow OWL class description atoms, OWL data range atoms, OWL datatype property atoms, sameAs and differentFrom atoms or built-in atoms. Not allowing OWL class description atoms is not a restriction since, for every class description C, we can always introduce an atomic named class $A_C \sqsubseteq C$ to the terminology of the KB and use A_C in the rule [MSS04]. The *sameAs*() and *differentFrom*() atoms are for syntactic convenience and not for increasing the expressivity of OWL. Such (in)equalities can already be expressed using the combined power of OWL and rules without explicit (in)equality atoms [HPB+04].

1.5.2 Semantics

A SWRL-like rule is treated as a *trigger rule* (see Section 1.2), as opposed to a SWRL rule which is a logical implication. So the contraposition law does not hold on a SWRL-like rule.

Operational semantics are given for these rules. The intended meaning for a SWRL-like rule is that: if the conditions in the rule body are proved to hold, then derive that the condition in the rule head holds. The rules are applicable only in one direction, namely, from rule body to rule head.

The rules are treated with *active domain semantics*, i.e., the variables in rule atoms are bound only to explicitly named individuals in the ontology domain and not to anonymous individuals encountered during the proof.

[15] Rules with zero atoms in the head or zero atoms in the body are not considered interesting for adding expressivity to OWL.

[16] This is not a restriction on expressivity, since a rule with a conjunction of atoms in the head can be easily transformed to multiple rules each with a single-atom head [HPB+04].

1.6 Contribution of Thesis

In the thesis we consider a SWRL-like rule language that shares its syntax with the SWRL rule language, but has different semantics. For OWL ontologies extended with rules expressed in this language, we provide a *DL based* reasoning support. Our contributions in this work include:

1. Recognizing the rule language, SWRL-like, for a practical implementation, which is very similar to the SWRL but differing slightly in semantics.

2. A proposal for a rule engine, called the **Fire** system, for reasoning with OWL ontologies extended with SWRL-like rules.

3. A prototype implementation of the Fire system offering sound, terminating and direct DL based reasoning that is synchronous with OWL reasoning.

4. Implementation of the *RETE algorithm* [For82] for handling pattern matching of DL facts and DL rule patterns.

1.7 Organization of Thesis

In Chapter 2 we present the background for this work. First, we state the background that guided the way we approached the problem addressed in the thesis. This is followed by a detailed description of the SWRL-like rule language, the rule reasoning process, possible approaches to ensure termination of reasoning and a sound and complete reasoning method. The chapter ends with a statement of the problem addressed and the solution we propose.

Chapter 3 is dedicated to the design of the proposed system **Fire** and the implementation details of the prototype developed by this thesis. The explanation is based on the overall architecture and the implemented architecture of Fire. At the end, we present a specific optimization we implemented for the prototype.

In Chapter 4 we discuss the results from our experiments with the prototype implementation. This is followed by an evaluation of the design and prototype implementation, by discussing how the example cases introduced are handled. We end this chapter with a discussion of related work.

In Chapter 5 we conclude the thesis with a discussion of our goals and achievements in the thesis followed by an overview of the future work.

Chapter 2

Preliminaries

This work is closely related to the dedicated state-of-the-art DL reasoner RACER [HMW04], which offers sound and complete reasoning for OWL DL almost completely[1]. In the view of bringing DL ontologies closer to the implemented systems (frame-based systems or rule based systems), our concern is towards extending DL ontologies with DL based rules, and supporting it by a DL based system for reasoning with rules. With this background, the standardization of OWL [OWL04] as the W3C standard for representing ontologies on the Web and the recent proposal to the W3C, SWRL [HPB+04], to extend OWL with rules, we were motivated towards implementing a DL based rule reasoning support for OWL ontologies extended with this such rules. A survey of the relevant literature in this direction shows significant amount of work done in combining ontologies and rules. However, for the present work we are concerned with the literature about combining *DL based* ontologies and rules. We find that most often the literature in this direction is combining with DL ontologies, the rules from either a Datalog (or its extensions) program, DL-safe subset of SWRL language or epistemic rules using the K-operator. A discussion of the related work is presented in Section 4.3.

Our research approach is to have OWL DL as the base and extend it with a rule language similar to SWRL, which can be supported by a practical implementation. This way we intend to still offer reasoning support for complete OWL DL, not restricting it in any way, and provide a practical reasoning support for rules offering direct inferencing over DL

[1] RACER's support for complete OWL DL currently has two limitations. One, approximation of nominals (individuals in class expressions) and two, currently not handling user-defined datatypes types given as external XML Schema specifications. See http://www.racer-systems.com/products/racerpro/features.phtml for details.

based rules. We consider a language similar to SWRL, which we call as the *SWRL-like* rule language, presented in Section 1.5. Our analysis and decisions are guided by two factors, (i) feasible practical implementation using existing tools and (ii) maintain decidability of the reasoning support for the combination.

2.1 SWRL-like Rules

We recall that the SWRL-like rules have a similar syntax as SWRL (see Sections 1.4 and 1.5). A rule is treated as a *trigger rule*, meaning the reasoning mechanism applies the rule in only one direction namely, antecedent to consequent. A typical SWRL-like rule in the human-readable syntax is of the form

$$antecedent \Rightarrow consequent$$

where the *antecedent* is made up of one or more rule *atoms* that are treated as a conjunction and the *consequent* is made up of exactly one rule *atom.* A rule *atom* can be either of

- Concept atom: A rule atom with a concept predicate of the form $C(X)$, where
  - C is an *OWL named class*
  - X is either a variable[2] or an OWL individual.
- Role atom: A rule atom with a role predicate of the form $R(X,Y)$, where
  - R is an *OWL object property*
  - X, Y are either variables or OWL individuals.

Unlike in SWRL, the SWRL-like rule language does not allow built-in relations, data ranges, sameAs, or differentFrom predicates. As mentioned in [HPB+04], not allowing sameAs and differentFrom kind of rule atoms does not necessarily decrease the expressive power of the combined language as these (in)equalities can still be expressed using the combined power of OWL and rules without these explicit atoms. Similarly allowing only *named* classes in concept atoms does not restrict expressivity in rules, since class descriptions can be replaced with an equivalent definition of a named concept added to the KB.

[2] In the thesis we use the standard convention of indicating a variable by prefixing it with a question mark e.g., $?x$

In many DL systems like CLASSIC[3] the concept of *epistemic K-operator* was used to provide a formal semantics for the procedural behavior of trigger rules [DLN+98]. Unlike the usual DL operators that refer to objects in the domain, the epistemic operator K refers to what the KB *knows* about the domain. In a sense, epistemic operators allow forms of local closed-world reasoning over an otherwise open-world knowledge.

2.2 The Rule Reasoning Process

In terms of the operational semantics of the SWRL-like rule language, the behavior of trigger rules in the rule reasoning process is usually described in terms of a forward reasoning process. In the process described below, we consider the following form of a typical trigger rule as a representative for SWRL-like rules for clarity.

$$C(X) \Rightarrow \mathcal{D}(X)$$

where C and $\mathcal{D}$ are concepts, and X is either a variable or an OWL individual. Consider a typical ontology knowledge base (KB) as a pair $\mathcal{K} = (\mathcal{T}, \mathcal{A})$, where $\mathcal{T}$ is the TBox (for terminologies) and $\mathcal{A}$ is the ABox (for assertions). This KB is extended by a finite set $\mathcal{R}$ of trigger rules typically of the form introduced above.

The operational semantics of the DL based trigger rules in $\mathcal{R}$ can be described by the forward reasoning process as follows [BN03]. Starting with an initial KB $\mathcal{K}$, a series of KBs $\mathcal{K}^{(0)}$, $\mathcal{K}^{(1)}$,... is constructed, where $\mathcal{K}^{(0)} = \mathcal{K}$ and $\mathcal{K}^{(i+1)}$ is obtained from $\mathcal{K}^{(i)}$ by adding a new assertion $\mathcal{D}(a)$ whenever $\mathcal{R}$ contains a rule $C(X) \Rightarrow \mathcal{D}(X)$, such that $\mathcal{K}^{(i)} \models C(a)$ holds, but $\mathcal{K}^{(i)}$ does not contain the fact $\mathcal{D}(a)$, for any individual a in $\mathcal{K}^{(i)}$. Note that this reasoning process eventually halts because the given initial KB $\mathcal{K}$ contains only finitely many individuals and there are only finitely many rules in given $\mathcal{R}$. Hence, there are only finitely many assertions $\mathcal{D}(a)$ that can possibly be added. The KB resulting after the series of rule applications is $\mathcal{K}^{(i)}$ having the same TBox as $\mathcal{K}^{(0)}$ but the ABox has been augmented by the assertions introduced by the rules. $\mathcal{K}^{(i)}$ is the final KB that results when no more augmentation to the KB is possible. Note that like OWL entailments, rule inferences are also monotonic, i.e., the assertions introduced by rules can only add new facts to the KB but never *take away* the already existing facts. Rules can only introduce new concept/role assertions and cannot introduce new individuals or new rules for that matter.

[3]http://www.research.att.com/sw/tools/classic/papers/manual.ps

The resulting final KB $\mathcal{K}^{(i)}$ is therefore independent of the order of rule applications. The process can be similarly extended for rules with property atoms also.

2.3 Consequences of Rule Application

The application of a SWRL-like rule augments the KB by introducing new assertions in it. For a rule engine it is important to consider the consequences of such augmentation in the KB.

In a given state of the KB, the body of a rule must be *satisfied* according to its semantics (see Section 1.4), for a rule to become *applicable* (or to be triggered) with an *instantiation*. Note that more than one such instantiation could trigger the rule simultaneously. Identifying such *triggered rules* and their corresponding *instantiation(s)* is one of the most important tasks in the rule reasoning process.

The *application* of such a triggered rule is done by *materializing* its head with the rule instantiation, i.e., by suitably *grounding* the variables in the rule head atom with bindings from the rule instantiation and then asserting the grounded atom to the KB. The materialized head of a triggered rule is called a *rule inference*. The rule is then said to be *fired* once with an instantiation. The following consequences of rule firing are vital to our discussion.

- A rule inference could be a part of the instantiation that in-turn triggers some rule in the next reasoning cycle.
- A rule inference could in-turn cause new entailments in the KB.
- A rule inference could contradict the facts in the KB, which we term as *introducing an inconsistency* in the KB.

These consequences have to be handled by the reasoning support, if the tight integration with OWL is to be retained. Meaning, given a state of the KB, the rule inferences and the possibly resulting new entailments in that state of the KB must be considered for the next cycle of reasoning. A *cycle* of reasoning comprises of identifying which rules are triggered in a given state of the KB and applying those rules to the KB according to some specific strategy (see Section 3.1.4). Also, an *inconsistency* possibly introduced in a reasoning cycle has to be detected at the earliest to enable the user or the application to recognize the corresponding rule that is responsible for introducing contradicting facts in the KB.

2.4 Decidability

OWL DL and rule languages like SWRL (whose rules are restricted forms of function-free Horn clauses) are both decidable logics. Their combination gives the much needed expressive power, at the same time makes the combined logic undecidable. From the illustrated example and discussions in [MSS04], we understand that one of the main reasons for this undecidability is because these rules allow reasoning over anonymous individuals encountered during the proof. This kind of arbitrary interaction between the OWL DL features and rule features is avoided, to maintain decidability, in the so-called *DL-safe* approach. The literature [MSS04] proposes this as a decidable rule extension to OWL DL and provides reasoning by reduction of the DL $\mathcal{SHIQ}$ to a disjunctive Datalog program.

2.4.1 DL-Safe Rules

In this approach, the atoms in the rules (a kind of Horn clauses) are restricted to being DL-atoms. That is, only unary atoms or binary atoms are allowed whose predicates can be respectively OWL classes or OWL properties. The OWL class used as a predicate must be a simple OWL named class. Further, it is required that each variable that occurs in the atoms of the rule body must occur in a non-DL atom in the body. This is to ensure that the variables are bound only to explicitly defined individuals in the KB and not to anonymous individuals encountered during the proof. For example:

$$hasParent(?x, ?y) \wedge hasBrother(?y, ?z) \Rightarrow hasUncle(?x, ?z)$$

This rule is *DL-unsafe* because the variables x, y and z in the body occur only in DL-atoms and not in non-DL atoms. This is made *DL-safe* as follows,

$$hasParent(?x, ?y) \wedge hasBrother(?y, ?z) \wedge O(?x) \wedge O(?y) \wedge O(?z) \Rightarrow hasUncle(?x, ?z)$$

by introducing an external non-DL predicate O in the rules. For every variable v occurring in the rule body, a non-DL atom, $O(v)$, is conjuncted to the rule body (in this case $O(x)$, $O(y)$ and $O(z)$). Also, for every explicit OWL individual $\imath$ in the KB, a concept assertion $O(\imath)$ is added to the KB. In this way the rule reasoning mechanism can ensure that the variables in the rule body are bound to only explicitly existing individuals in the KB. This is called DL-safety and it makes the reasoning approach *decidable*. However, it is important to consider the consequence of such a *safety* restriction on an example case illustrated

below. Consider the following DL-unsafe rule,

$$R(?x,?y) \wedge R(?z,?y) \Rightarrow C(?y)$$

where R is an object property, C is a named class and i, j are variables, along with the following assertions

$$anne : \exists S.\exists R.\{jim\}$$

$$anne : \exists U.\exists R.\{jim\}$$

The former assertion states that *anne* is an instance of a concept that is related by role S to some individual that is in-turn related by role R to the individual *jim*. Similarly, the latter states that *anne* is an instance of a concept that is related by role U to some individual that is in-turn related by role R to the individual *jim*. Intuitively, *jim* should be inferred as an instance of C from the DL-unsafe rule shown above, though *jim* is not related by property R to an *explicitly named* individual of the KB considered. But with the DL-safety restriction applied to the above rule, it is easy to see that the inference $j : C$ is not derived by rule reasoning. This kind of restricting the reasoning to explicitly named individuals in the KB is generally known in the DL world as *active domain semantics*. The advantage of retaining decidability in this way vs. a need for applications or users to be able to derive such an inference ($j : C$) is an issue of discussion on which an application-specific rule reasoning support should be based on. The authors of [HAMS05] are of the opinion that this kind of DL-safety restriction is not very restrictive in many cases because a significant number of applications like metadata management on the Semantic Web or information integration require extensive ABox reasoning (query answering or instance checking) and such applications know most individuals by name. But for applications that require intensional (TBox) reasoning, this is obviously a severe restriction as typically only a handful of individuals is known by name in such applications.

The DL reasoner RACER and its query language nRQL both implement this *active domain semantics* in their ABox query services. So, any system making use of inference services of RACER automatically inherits this *safety* feature.

2.5 Sound and Complete Reasoning

A sound and complete reasoning is both sensible and always desirable by any application. Providing a sound and complete reasoning is hence a main focus of a reasoning support for

a combination of OWL and rules. In this context we present the approach followed in the CARIN system [LR98] and later present how we follow a similar approach in our current work.

The CARIN system is a family of representation languages providing a hybrid integration of Horn rules and description logics (specifically, subsets of the $\mathcal{ALCNR}$ DL). Two specific subsets of CARIN are identified in [LR98] and a sound and complete reasoning procedure is offered for both. The reasoning is done in two steps namely, the DL-reasoning step and the rule reasoning step. This procedure is well summarized in [ADG+05] as follows:

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

Following this approach in CARIN, we make a conjecture that a similar procedure described below is sound and complete for OWL ontologies extended with SWRL-like rules. The procedure is also done in two steps as follows. The tableau completion rules are used to construct all possible *completions* of the KB. Each clash-free completion will be represented by a finite set of facts (concept and role assertions), and determines a *canonical model* of the KB. Note that there can be finitely many such completions. In each such clash-free completion, a forward chaining is performed with its facts and *all* the SWRL-rules of the KB. The resulting instantiations for the rule head of applicable rules are derived as the set of *rule inferences* from each completion. The intersection of all such sets derived from each completion, determines the set of rule inferences that are common in all models of the KB. This intersection set determines the final set of rule inferences for the given state of the KB. A concept assertion $C(a)$ where C is an OWL concept and a is an OWL individual, belonging to this final set, is entailed by the OWL KB iff it is inferred as a rule inference in the forward chaining on each of the clash-free completions of the KB. The *final set* of rule

inferences is collectively asserted in the KB and any entailments caused as a consequence of this augmentation will be considered when the completions are constructed for the next cycle of rule reasoning. If any of the rule inferences contradicts the facts present in the KB, then they are not asserted and the reasoning process stops. If there are no contradictions, then the reasoning process stops if no *new* inferences could be derived in the current reasoning cycle on the given state of the KB.

2.6 Problem Definition and the Proposed Solution

We conclude this chapter by defining the problem we address in this work and our proposed solution. In the research community of the Semantic Web and description logics, there have been growing efforts to extend the OWL language for Web ontologies with rules, for the much needed expressive power offered by the combination. In this context we propose an extension to OWL with a *DL based* rule language called the SWRL-like rule language (discussed in Sections 1.5 and 2.1). The preliminaries needed for a reasoning support for OWL ontologies extended with SWRL-like rules, were presented in this chapter. The proposed rule language extension has the operational semantics of trigger rules (Section 2.2). The reasoning support for the proposed extension follows the *active domain semantics* to ensure termination. The rule reasoning procedure (Section 2.5) is conjectured to be sound and complete based on the approach followed in the CARIN system [LR98], of reasoning over all completions of a KB.

The discussions presented in this chapter lay a foundation for the reasoning engine **Fire**, proposed in the thesis. It is a DL based rule engine to reason with OWL ontologies extended with SWRL-like rules. The design and implementation of a prototype of Fire is presented in the next chapter.

Chapter 3

Fire – Design and Implementation of the Prototype

Previously, in Chapter 2 we discussed the syntax and semantics of the SWRL-like rule language. We also discussed the rule reasoning process and a procedure for ensuring sound and complete reasoning. The discussions lay a foundation for the topic of this chapter. In this chapter we present the design of the proposed rule engine, **Fire**, and implementation of its prototype. First, the design and the overall system architecture are presented. Then, the details of implementation and decisions made are presented.

3.1 Design

The essential function of the proposed rule engine is to *fire* rules, i.e., to identify rule instantiations that trigger rules and to apply those triggered rules. Hence the rule engine is aptly named **Fire**. The overall system architecture in the design phase is shown in Figure 2. The Fire system is designed as a *stand-alone* reasoning engine that could be used either as a rule reasoning extension to a DL reasoner, or as a rule reasoning component of an existing Sematic Web (or Web inference) application.

The shaded area in Figure 2 represents the Fire system. The figure shows different components of Fire and its interaction with the external world. All aspects of the design are discussed from the four perspectives covered in Sections 3.1.1 to 3.1.4.

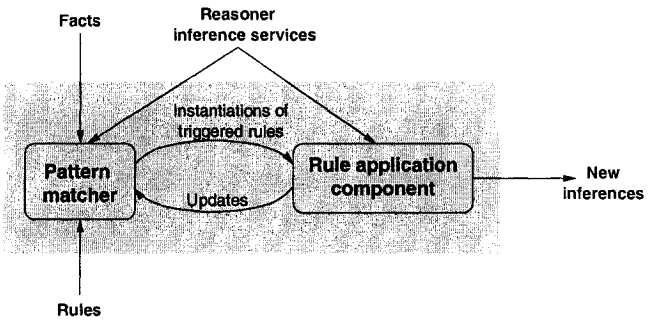


Figure 2: The design phase architecture of the Fire system.

3.1.1 DL Reasoner Services

The Fire system is a *homogeneous DL based* rule engine that is closely integrated with a DL reasoner. Reasoning services of a DL reasoner are very essential in the working of Fire. The reasoner services required are:

- *ABox consistency checking* to identify whenever an *inconsistency* is introduced as a consequence of rule application,
- *Instance retrieval* with active domain semantics, to obtain concept/role instances that could trigger rules,
- *ABox cloning* to verify the consequence of rule application on the ABox before committing changes in the ABox.

The Fire system can integrate with any of the existing DL reasoners offering the above services. However for efficiency reasons, it must be ensured that there is not much processing overhead in communicating with the reasoner.

3.1.2 Reading Rules

Rules defined in the ontology must be *read into* Fire. Most of the DL reasoners available provide a parser for reading-in the OWL ontology. If no parser is available to read the SWRL-like rules, then this can be accomplished either by implementing a rule parser or by making use of an existing third party rule parser. This is depicted in Figure 2 by the *Rules* input to the Fire system.

3.1.3 Pattern Matching

A rule engine deals with a system having a fixed number of rules and a KB that grows by augmentation due to the application rules. Pattern matching in this case is concerned with matching the DL facts (concept and role assertions) in the KB with the patterns (concept or role atoms) in the rule bodies. A concept assertion $Female(suzanne)$ is said to be a *match* for a pattern $Female(?x)$ appearing as a concept atom in a rule body. A set of such matches for all the patterns in a rule body form an *instantiation* of the rule and this set provides bindings for variables in the rule head. These bindings instantiate the rule head and are used in applying the rule. An obviously naïve algorithm to pattern matching in this case would be to match each fact (asserted, entailed or inferred) in the KB against each rule pattern. Generally, the number of new inferences derived is often very small compared to the number of facts initially considered for the matching process. Due to this, in this approach most of the matches obtained in a cycle are identical to the matches obtained in the previous cycles. Therefore the algorithm can be very inefficient for ontologies with huge number of rules and facts, as most of the processing time will be wasted in repeating useless comparisons.

In order to overcome this problem, an intelligent approach must (i) reuse matches from earlier cycles, (ii) avoid repeated comparisons, (iii) be informed about the initial facts and focus on newly inferred facts for matching and (iv)identify instantiations that trigger rules without having to try each rule sequentially. These requirements aptly fit the description of the original RETE algorithm [For82] that forms the basis of the popular expert system OPS5 (Official Production System) amongst others. This algorithm speeds up the matching process by constructing a network of nodes based on information about the patterns in the body of rules. Matching is performed once initially when the network is constructed and this information is stored in the network. As the KB augments with each rule reasoning cycle, the algorithm looks only for *changes* (newly inferred facts) in the KB. The original RETE algorithm is well explained in [GR98]. This can be implemented to match DL facts and patterns for pattern matching in Fire. Given a finite set of rules and an initial state of the KB, the pattern matching algorithm basically works on all the given rules, identifies applicable ones and provides bindings to instantiate the head of applicable rules. The efficiency of the RETE algorithm is based on the assumption that compared to the number of facts in the initial collection for pattern matching, the number of new facts added is often very small. That is the facts change slowly over time. This is true in case of

OWL ontologies extended with SWRL-like rules.

The prototype of Fire implemented in the thesis implements the RETE algorithm for the pattern matcher component. It is described in detail in Section 3.2.3. Because of the nature and working of the RETE algorithm, the rules that become applicable in a given state of the KB and their corresponding instantiations are provided by the RETE pattern matcher all at once.

3.1.4 Rule Application

The instantiations for rule heads obtained from the pattern matching are *materialized* in the KB by asserting them as new facts. This is termed rule application. Fire supports *monotonic* reasoning, since the SWRL-like rules are allowed only to make new assertions about existing individuals in the OWL ontology. Existing DL facts cannot be retracted as a rule consequence and neither can new individuals be introduced in the KB. A rule once *fired* will become applicable again in subsequent pattern matching cycles since the facts that trigger the rule are always present in the KB. This would result in having to infer a fact that already exists in the KB, which has to be avoided. A dedicated *rule application component* in Fire's architecture in Figure 2 must detect and avoid asserting such duplicate inferences in the KB.

Consistency of the ontology KB is of prime importance in our design approach of Fire. If the instantiations for rule heads obtained from pattern matching, happen to contradict the facts in the KB, then the *rule application component* does not materialize this set of instantiations and terminates the Fire rule engine. This is to enable the user or the application to identify the rule that is responsible for introducing facts that contradict the KB. To detect contradictions, the rule application component makes use of the ABox cloning and ABox consistency checking services from the DL reasoner. In case of no contradictions, the component proceeds normally with the materialization. New entailments that might hold in the KB as a consequence of materialized new rule inferences are collected as *updates*. These updates are provided to the pattern matcher which proceeds with the pattern matching for the next *cycle* of rule reasoning.

Fire is designed to operate in two modes based on the two modes of operation of its rule application component. They are:

- **Default Mode:** When Fire operates in this mode, the rule application component performs a check on the *final set of rule inferences* obtained in the reasoning cycle to

test if some rule inference contradicts the facts in the KB. On detecting a contradiction, this pre-commit consistency check of the KB drives the system to termination. Fire terminates in such a situation because reasoning with an inconsistent KB is not interesting. Detecting the rule instantiation that resulted in the contradicting rule inference, as early as possible, is an important requirement for the user or the application. To allow this, the *debugging mode* of operation is provided.

- **Debugging Mode:** When Fire operates in this mode, the pre-commit consistency check of the rule application component is replaced by a *one-at-a-time* assertion of the rule inferences in the final set. This means an expensive pre-commit consistency check for each rule inference in the final set of inferences in the reasoning cycle. On detecting a contradiction in the KB, the debugging mode terminates the reasoning without committing such an inference and the user or the application is allowed to intervene and handle the consequence. Reasoning in this mode would be very helpful in detecting at the earliest, any rule inference (hence the specific instantiation that triggered the rule) that is the cause for introducing a contradition in the KB. Note that in the worst case where such an inference is the last of the final set of inferences, discovering the contradiction will obviously be delayed till the end.

Comparatively, for a given state of the KB, the debugging mode is obviously slower as the consistency-check of the KB is done once for *every* rule inference in the final set. Consistency checking is a costly (time consuming) service from the DL reasoner. This helps localize the cause of *inconsistency* to the rule instantiation that caused it. The default mode is comparatively faster as the consistency-check is done once for the *entire* final set of rule inferences in a cycle. The cause of inconsistency is localized here to a reasoning cycle. A contradiction possibly introduced in the earlier stages of the cycle goes undetected until all rule inferences in that cycle are obtained. To pinpoint the exact rule instantiation that results in the contradicting inference might then become expensive in the end of the cycle.

In either modes of operation of Fire, the order of rule application does not affect the *final* result (also see Section 2.2), since the reasoning stops on encountering an *inconsistency* causing rule inference. Hence, specialized conflict resolution strategies like depth, breadth, etc. offered by the CLIPS[1] system, or the LEX, MEA, etc. offered by the OPS5 production system, are not offered in Fire.

[1] http://www.ghg.net/clips/download/documentation/bpg.pdf

3.1.5 Termination

The reasoning process of Fire terminates under two conditions as follows:

- When the rule inferences derived in the current reasoning cycle contradict the state of the KB
- When the rule application component ensures that no *new* rule inferences were derived in the current reasoning cycle

Under either condition, the Fire system terminates by providing the external application with the set of all new inferences that were materialized by the system since the first reasoning cycle.

3.1.6 Reasoning Pseudo-algorithm

Here we present a pseudo-algorithm describing the reasoning process in Fire. It highlights the rule application process. The pattern matching process is not outlined here as its implementation details are described in Sections 3.2.3 and 3.2.5.

```
Subroutine FIRE_REASON(patterns,facts)
   applied_inferences = empty
   DO
      inferences = PATTERN_MATCH(patterns,facts)
      updates = empty
      terminate = RULE_APPLY(inferences,facts,applied_inferences,updates)
      facts = add_to_collection(facts,updates)
   UNTIL terminate is true
   return applied_inferences
   EXIT FIRE_REASON
END FIRE_REASON

Subroutine RULE_APPLY(inferences,applied_inferences)
   terminate = false
   new_inferences = get_new_inferences(inferences)
   IF new_inferences is empty THEN
      terminate = true
```

```
        RETURN terminate
    ELSE
        FOR i = start TO end of new_inferences
            IF contradict_with_KB(i) THEN
                terminate = true
                RETURN terminate
            ELSE
                inferences_to_apply = add_to_collection(inferences_to_apply,i)
            END IF
        END FOR
    END IF
    updates = apply(inferences_to_apply)
    applied_inferences = add_to_collection(applied_inferences,inferences_to_apply)
END RULE_APPLY
```

The subroutine FIRE_REASON outlines the steps in the overall reasoning process of Fire. The PATTERN_MATCH and RULE_APPLY are calls to the pattern matcher component and the rule application component of Fire respectively. The RULE_APPLY subroutine signals termination under two conditions: (i) no *new* inference derived, and (ii) some new inference contradicts the facts in the KB. The FIRE_REASON subroutine terminates the reasoning process when the RULE_APPLY subroutine identifies these conditions and signals termination.

3.1.7 Soundness, Completeness, and Termination of Our Design

The Fire system is designed to be tightly integrated with a DL reasoner. Provided soundness is guaranteed by the DL reasoner, the rule reasoning offered by the Fire system is also guaranteed to be sound because of the tight integration. We conjecture that the reasoning in Fire is complete if the proposed approach (see Section 2.5) of rule reasoning over each of the clash-free completions of the ontology KB is implemented. Termination of reasoning is ensured by applying rules to only explicitly defined individuals in the ontology KB. This kind of reasoning with active domain semantics is essentially inherited by the DL reasoner chosen in the implemented system.

3.2 Implementation

The design of the **Fire** system described in Section 3.1 is implemented by the current prototype, in Java. The supported rules are from the SWRL-like rule language (see Sections 1.5 and 2.1 for the rule syntax and which forms of rule atoms are permitted by the rule language). The overall design phase architecture of the Fire system shown in Figure 2 is refined in the implementation phase shown in Figure 3.

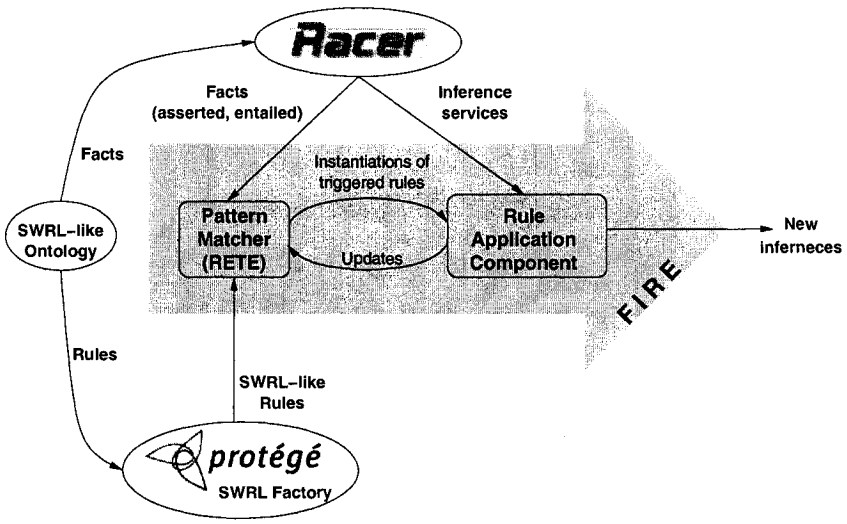


Figure 3: The refined architecture of the Fire system in the implementation phase.

The shaded arrow-shaped area represents the implemented prototype of Fire. The refinements from the design phase are clearly shown. Similar to the discussion in the design phase, the aspects of the implementation are also discussed in terms of the same four perspectives in the following Sections 3.2.1 to 3.2.4.

3.2.1 DL Reasoner

Our obvious choice for the reasoner is the RACER DL reasoner [HM01, HMW04] because of our background. Some of the implementation details and decisions are hence customized to RACER and its services. However, in principle any of the existing DL reasoners could be chosen. The prototype of Fire uses the JRacer [HMW04] Java API to communicate with the RACER system via a TCP connection. The prototype also uses the nRQL (new Racer Query Language) [HMW04] from RACER to run instance retrieval queries. The JRacer API has a convenient mapping of most of RACER's and nRQL's commands to Java classes.

The OWL part of the ontology is parsed using RACER's internal parsing mechanism. The *facts* in the ontology KB needed for the pattern matcher are obtained from instance retrieval nRQL queries to the RACER system. This tight integration with the DL reasoner ensures that all facts entailed by the OWL part of the ontology (asserted and entailed facts) are considered for rule reasoning. RACER already implements the *active domain semantics* which is followed by Fire (see Section 1.5). The role and concept instances obtained as answer sets to nRQL queries, used for binding rule variables thus ensure the fact that variables are bound only to explicit individuals in the ABox and not the anonymous ones encountered during the proof.

3.2.2 Rule Parser

Fire needs Java output from a SWRL-like rule parser. A third party SWRL rule parser is available from the Protégé ontology editor, called the Protégé SWRL Factory[2] which serves our purpose. It is used in Fire since SWRL-like rules share the same syntax as SWRL. It provides a Java API for easy access of rules with SWRL syntax, directly in the DL form and not any intermediate translated form. Fire basically reads an ontology file with .owl file extension, which contains both the OWL part and the SWRL-like rule part of the ontology.

3.2.3 Pattern Matching Component

The pattern matcher component of Fire implements the RETE algorithm for matching DL patterns in SWRL-like rules with the DL facts in the ontology. The implementation of this component of Fire is presented in detail in Section 3.2.5.

3.2.4 Rule Application Component

The rule application component of Fire detects and avoids materializing repeated rule inferences as discussed in the design (section 3.1). It proceeds with materialization of only *new* inferences. If there are no new inferences, the Fire system terminates providing the set of materialized inferences so far, since the first rule reasoning cycle. A set of new inferences derived in the current reasoning cycle are materialized in the reasoner's ABox (asserted) simultaneously as a batch, provided they do not introduce an *inconsistency* in the current

[2]Details of the Protégé SWRL Factory can be found from the URL http://protege.stanford.edu/plugins/owl/swrl/SWRLFactory.html

state of the ABox. An *inconsistency* introduced by this set of new inferences also drives the system to termination, providing the set of materialized inferences so far until the previous reasoning cycle. In case *inconsistencies* are not introduced, any new entailments that hold in the ABox as a result of new materialization are obtained and passed to the pattern matcher component as *updates* for its facts collection. This starts a new reasoning cycle in the pattern matcher. Inclusion of these updates in the next reasoning cycle ensures that rule reasoning is synchronous with the OWL reasoning of the DL reasoner.

3.2.5 Implementation of the RETE Algorithm

The RETE algorithm [For82] is implemented for the pattern matcher component in Fire, to match DL facts with DL rule patterns. This algorithm was chosen instead of a simple naïve matching algorithm because of its speed and reuse of match results. It proves to be efficient in case of repeated matching cycles which a system like Fire encounters. An experimental evaluation of this algorithm vs. a naïve algorithm is presented in Section 4.1. The implemented RETE algorithm can be understood by dividing it into two major steps: (i) construction of the network and (ii) the matching process.

Construction of the Network

The algorithm begins by constructing a network of *nodes* based on the *patterns* in the rules of the ontology. Patterns here are nothing but rule atoms, concept or role atoms. The rules are *processed* for its patterns (i.e., the patterns are identified from the rule body and head) and a part of the network structure is constructed for each rule processed. The parts are appended to one another as more rules are processed. When all the rules are thus processed, the entire RETE network is ready. Figure 4 shows the RETE network constructed for the ontology example in Figure 5. The network consists of four types of nodes:

- Root Node: It is a single node forming the root of the network and the entry point for the facts of the KB into the network.
- A-Nodes: They have one input and are called SELECT Alpha nodes. Basically, for each pattern in a rule body, a corresponding A-Node is added in the network, e.g., the node $A1$ in Figure 4.
- B-Nodes: They have two inputs and are called JOIN Beta nodes. The conjunction of two consequent atoms in a rule body is represented by a B-Node which joins two

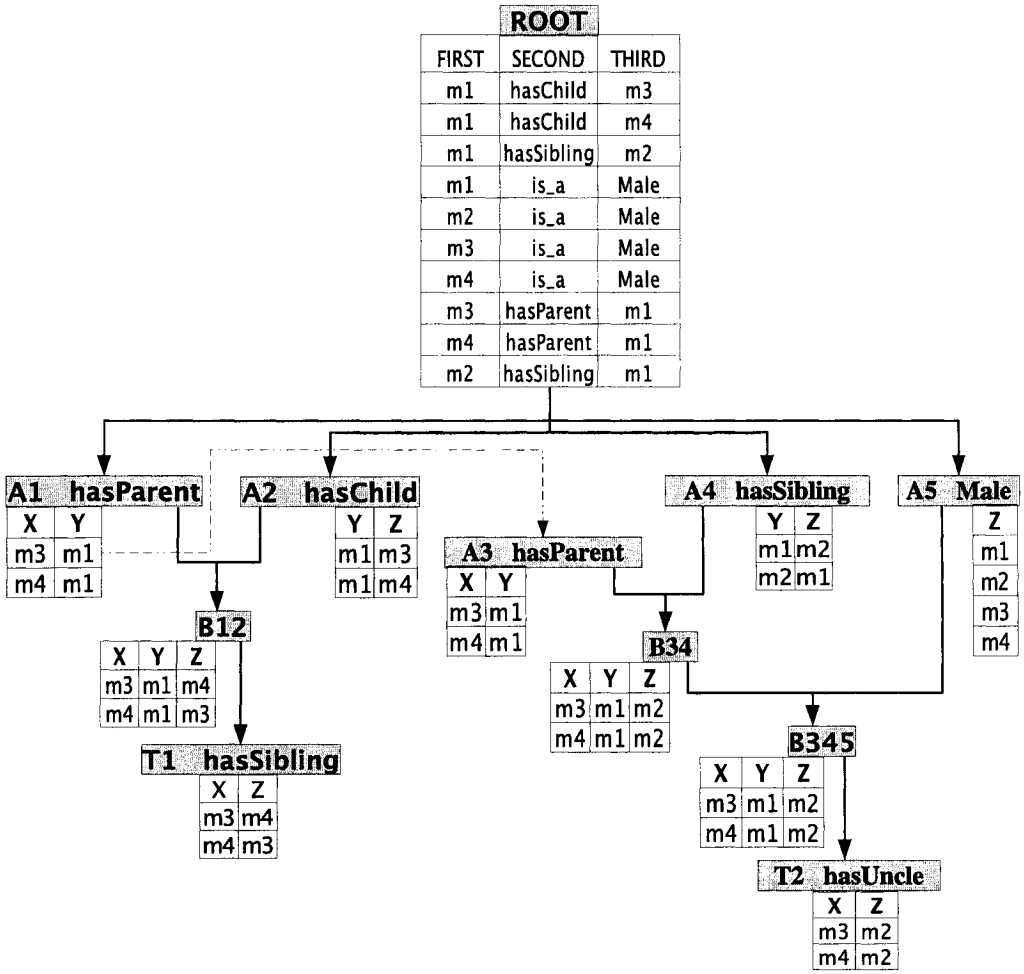


Figure 4: Illustration of the RETE algorithm for pattern matching.

input A-Nodes (which represent the conjunct atoms), e.g., the node $B12$ joining the nodes $A1$ and $A2$ in Figure 4.

- T-Nodes: They have one input and are called PROJECT Terminal nodes. There is a T-Node for each rule head atom, e.g., the node $T1$ in Figure 4.

The portion of the network in Figure 4 represented by the nodes $A1$, $A2$, $B12$ and $T1$ correspond to the *structure* of the first rule in the ontology example in Figure 5. Similarly, the portion of the network represented by the nodes $A3$, $A4$, $A5$, $B34$, $B345$ and $T2$ in Figure 4 correspond to the structure of the second rule in the example in Figure 5. Each node in the network has a *relation* associated with it and the *records* in these relations represent facts from the KB that match the node's pattern. Henceforth, a relation will be

referred to as a *table* and a record in a table as a *row*. Rows of the Root-Node table represent at any time those facts in the KB that are not yet selected as *matches* in the pattern matching process. A-Node table rows are the result of a relational SELECT[3] operation on the Root-Node table. B-Node table rows are results of a relational JOIN[4] operation on tables of its two input nodes (left input and right input). T-Node table rows are a result of a relational PROJECT operation[5] on its input B-Node table. Structurally, the Root-Node is at the top with A-Nodes directly below. A-Nodes are followed by zero or more *levels* of B-Nodes[6] and below them are T-Nodes that form the *leaves* of the network. A-Nodes accomplish the task of matching KB facts against the rule patterns they represent. B-Nodes accomplish the task of carrying these matches across patterns of a rule body. Their rows are obtained by joining tables of their input nodes. A T-Node represents the atomic head of a rule. Only those columns required for binding the variables in the head of such a rule are *projected* from the table of the T-Node's input node. There is one T-Node in the network for every rule in the KB. With the exception of the Root Node, each node in the network represents information about rule patterns. At each such node we store information about facts that match the patterns in nodes from Root down to and including this node.

Matching Process

Each fact in the KB is *fed* into the RETE network in terms of a row in the Root-Node table. Any row entering the Root-Node table is sent to *all* the A-Nodes where a SELECT operation is done to *filter* only rows matching the pattern associated with the A-Node. Appropriate rows are filtered down to B-Nodes if they satisfy the JOIN condition across A-Node patterns, and finally to T-Nodes if they are chosen by the PROJECT operation. When a set of facts passes all the way from the Root down to a T-Node, it has passed the matching test for a rule body and represents an instantiation of the rule body. For instance, the individuals $m3$, $m1$ and $m4$ (first row in node $B12$ table) as substitutions for the variables

[3]A relational SELECT operation obtains a subset of tuples of a relation (here rows of a table) that satisfy the SELECT condition (here the condition is the row matching the pattern associated with the A-Node).

[4]A relational JOIN operation combines related tuples (rows) from two relations (tables). Specifically, we use a natural JOIN operation which computes the cartesian product of the two tables, selects from the product only those rows having equal values for some common attribute, and removes the duplicate attribute from the resulting table.

[5]A relational PROJECT selects a subset of the attributes of a relation by specifying the names of the required attribute.

[6]A rule with a single atom in the body does not need a corresponding B-Node in the RETE network. In this case, an A-Node is directly followed by a T-Node and hence *zero* levels of B-Nodes.

x, y and z respectively, reaching the node $B12$ represent an instantiation for the body of the first rule in the example ontology in Figure 5. This instantiation reaching node $T1$ triggers the rule represented by that T-Node (in this case the first rule in the example ontology in Figure 5). The first row of the table of node $T1$, i.e., $m3$ and $m4$ for variables x and z respectively, are the required bindings to materialize the atomic head of the rule (1) in the example ontology. This is just one instantiation that triggered the rule. In the example we see that there are in total two instantiations that trigger the rule represented by node $T1$ whose table correspondingly has two rows. All the instantiations of all triggered rules obtained from the matching process are provided to the rule application component, where they are applied. This one cycle of reasoning or *one cycle of firing* of rules may result in new entailments in the ontology KB. These *changes* are provided back to the pattern matcher by the rule application component as *updates* that are *fed* into the RETE network through Root-node. The next reasoning cycle begins by starting the matching process again.

3.2.6 Soundness, Completeness, and Termination of the Prototype

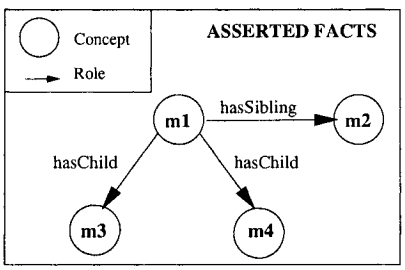
The implemented prototype of Fire tightly integrated with the DL reasoner RACER, which guarantees sound reasoning services. The query answering service from the nRQL query language subsystem of RACER is also sound and it implements the active domain semantics. The current prototype of Fire inherits these features from RACER and nRQL. Hence, the reasoning offered in this prototype is sound and terminating. Currently, neither RACER nor any of the other existing state-of-the-art DL reasoners offers the services to compute all clash-free completions of an OWL KB and to reason over each such completion. Implementing these reasoner services is non-trivial and time-consuming. Discussions with the team behind the RACER reasoner indicated that such services could not be provided by RACER within the time frame of this thesis. Hence the reasoning approach proposed in the design is not implemented in this prototype of Fire. Consequently, completeness of reasoning is not guaranteed in this prototype. However, discussions have indicated that subsequent releases of the RACER system will provide these services in the near future. A corresponding implementation of Fire incorporating these services and providing a sound, complete and terminating (i.e., decidable) reasoning over OWL ontologies with rules, will soon follow.

3.3 Implemented Optimization

Generally, it is possible that patterns (concept or role atoms) in different rule bodies may have the "same predicates". Such patterns will have same *number* of variables, but might have different variable labels. For example, the role atom $hasParent(?x, ?y)$ in both the SWRL-like rules in the example ontology in Figure 5. Such similarity can be utilized to optimize the construction of the RETE network. *Similar* A-Nodes corresponding to such similar patterns can share the process of obtaining their table rows and avoid a SELECT operation on the Root table. This is implemented by having two levels of A-Nodes namely A-Node1 (that selects its rows from the Root table) and A-Node2 (that has a pattern similar to an earlier A-Node1, so directly takes its rows from that A-Node1). In the example in Figure 4 the nodes $A1$ and $A3$ share such a similarity. Node $A1$ is of the kind A-Node1 and node $A3$ is of the kind A-Node2. Node $A1$ was encountered earlier in the network construction step and hence any *similar* node encountered later is made to be of kind A-Node2. Note that the order of processing rules for network construction does not make a difference to the overall structure of the network. It also does not make a difference which of the *similar* nodes is created as A-Node1 and which as A-Node2, since the overall effect is that one SELECT operation on the Root table is shared by two or more similar nodes. In case of ontologies in which a lot of such similarities exist, *duplicate* SELECT operations that provide identical results are avoided and the overall speed of the RETE procedure could be optimized.

Concepts	$Male$
Roles	$hasChild, hasParent, hasSibling, hasUncle$
Restrictions	$hasSibling \equiv hasSibling^{-}$ $(symmetric\ role)$ $hasParent \equiv hasChild^{-}$ $(inverse\ roles)$
Asserted Facts	$Male : \{(m1), (m2), (m3), (m4)\}$ $hasChild : \{(m1, m3), (m1, m4)\}$ $hasSibling : \{(m1, m2)\}$
Entailed Facts	$hasParent : \{(m3, m1), (m4, m1)\}$ $hasSibling : \{(m2, m1)\}$
SWRL-like Rules	$hasParent(?x, ?y) \wedge hasChild(?y, ?z) \Rightarrow hasSibling(?x, ?z)$ $hasParent(?x, ?y) \wedge hasSibling(?y, ?z) \wedge Male(?z) \Rightarrow hasUncle(?x, ?z)$

(a) An example ontology showing family relations using SWRL-like rules.



(b) Pictorial representation of the asserted facts in the example ontology.

Figure 5: An example OWL ontology with SWRL-like rules.

Chapter 4

Evaluation

The current prototype of the **Fire** system does not implement the reasoning procedure described in Section 2.5. The reason is that none of the current state-of-the-art DL reasoners (including the RACER system used) provide a service for computing all *completions* of a KB and reasoning over each completion. To provide such a service is not a trivial task for a reasoner and it involves significant changes in the design of its core system. Implementing this reasoner service is out of the time frame and scope of the thesis. Hence, we are unable to provide a comprehensive evaluation of performance of Fire with its handling of typical test cases, at this point. Nontheless, in this chapter we provide a preliminary evaluation of Fire, from different perspectives.

First, we present the results of experiments we conducted with our RETE algorithm implementation and evaluate its performance. Then, with the aid of few example cases we evaluate how the prototype system handles these cases and where it fails. Finally, we conclude this chapter with a discussion of related work relevant to the line of research in the thesis and discuss their similarities and differences with our proposal.

4.1 Experimental Results

In the preliminary stages of the implementation of **Fire**, we implemented two algorithms for the pattern matcher component: a naïve pattern matching algorithm and the RETE algorithm (section 3.2.5). The naïve algorithm implemented obtains bindings for the rule head of applicable rules, by comparing every fact entailed by the given state of the KB with every pattern of every rule. The process starts with the first pattern in the body of a rule. A

match (or a set of matches) found for this pattern is carried over to the pattern matching for the next pattern in the rule body. Both the matches (or the set of matches) are *joined* to get the same effect as achieved by the JOIN nodes in the RETE network. A match (or a set of matches) thus *joined* is carried over similarly to the pattern matching of the next pattern in the rule body, and so on until the last atom in the rule body. The *combined match* (or the combined match set) at this stage represents the instantiation(s) of the rule body. This is (these are) used to bind the variables in the atomic rule head and obtain instantiation(s) for the rule head. Moreover, the entire process repeats in every new reasoning cycle. It is easy to see why this approach is inefficient in cases of large number of rules and large number of facts in the ontology, as most of the processing time gets wasted in repeating useless calculations. Nonetheless, we conducted experiments to compare the performances of our implementation of the RETE algorithm and the naïve algorithm, to see how fast the RETE performed. The obtained results are summarized in the Table 1.

No. of rules	**No. of initial facts**	**No. of fired inferences**	**Time with RETE (in sec)**	**Time with naïve (in sec)**
500	584	995	0.875	21.093
1100	1301	2486	3.813	115.468
1772	2127	3950	12.514	303.201
2280	2698	5085	25.031	524.155
2726	3229	6158	39.469	762.172

Table 1: Summary of comparison between the performances of the RETE algorithm approach and the naïve approach to pattern matching

Test cases were synthetically generated such that inferences fired by some rules trigger other rules. Rules were designed such that some rule inferences trigger other rules in a chain-like order and some other rule inferences trigger in a tree-like order. This test was only on the *pattern matcher* component of Fire, in this case using two pattern matching algorithms. Therefore, the rule application step was simulated by just *feeding-in* all the *new* inferences derived in one matching cycle as *updates* to the next cycle. Rules were designed not to introduce any contradictions in the KB. New inferences derived in each matching cycle were collected for the count of total number of newly inferred facts. The matching stops when no *new* inference is derived. The Table 1 shows the number of rules, number of initially asserted facts and number of totally inferred facts, along with the runtimes required by the RETE and naïve algorithms. These were obtained on a desktop computer with a 2.8 GHz Intel processor, 1GB of RAM and running Windows XP operating system. This test

was mainly conducted for measuring the performance of RETE vs. naïve approach and hence no DL reasoner was used. The calculated time includes only the processing time for the pattern matcher component. That is, time needed to (i) identify matches and rule triggers, (ii) feed the derived inferences to the fact store for the next cycle, and (iii) repeat steps i and ii until no new inference is possible. It does *not* include the time taken to *materialize* a fired inference or the time to load the program. The results indicate that the RETE approach substantially speeds up the pattern matching process and is more than 20 times faster than the naïve approach. This reassures employing the RETE algorithm for pattern matching in the Fire system.

4.2 Example Cases

In this section we present few examples to illustrate how the prototype Fire system handles different scenarios. The first three examples highlight the expressive power added to OWL by rules and to show the importance of Fire's tight integration with the inferences of a DL reasoner. Next we present two example scenarios and discuss when and why Fire cannot infer intended results when handling them.

4.2.1 Example1

We designed an ontology *family.owl* representing the usual family relationships. We used SWRL-like rules to define some relationships that were otherwise not expressible in OWL without rules. This ontology is described in detail in Section A.1 and a step-by-step explanation of its execution in Fire is presented in the Appendix A. For a clear understanding of the functioning of the prototype, we refer interested readers to the mentioned sections of the thesis. In this section we highlight some of the rules used in the example ontology *family.owl*. In particular, the following rules

$$hasParent(?x,?y) \wedge hasSister(?y,?z) \Rightarrow hasAunt(?x,?z)$$
$$hasChild(?x,?y) \wedge hasSpouse(?x,?z) \Rightarrow hasChild(?z,?y)$$
$$hasParent(?x,?y) \wedge hasChild(?y,?z) \Rightarrow hasSibling(?x,?z)$$
$$hasParent(?x,?y) \wedge hasBrother(?y,?z) \Rightarrow hasUncle(?x,?z)$$

are typical examples whose models easily lose the *tree property* (see Section 1.4.2). These forms cannot be expressed in OWL without rules. The current prototype of Fire handles

these kinds of rules. The reader is encouraged to refer to the Appendix A for a step-by-step explanation of executing this example ontology in Fire. An additional case with a contradiction-introducing rule is also discussed there, highlighting how Fire deals with *inconsistencies* introduced in the KB by rules.

4.2.2 Example 2

Consider an ontology with the following rule

$$hasLocation(?x, ?y) \wedge isPartOf(?y, ?z) \Rightarrow hasLocation(?x, ?z)$$

with $isPartOf$ being a transitive role. The rule expresses the idea an individual's property of *location* is propagated through the transitive role $isPartOf$. This is a typical idea that needs to be expressed in medical ontologies. Suppose this example ontology has assertions $(fracture, femur) : hasLocation$ and $(femur, leg) : isPartOf$, then the above rule becomes applicable and infers that the fracture is located in the leg, i.e., $(fracture, leg) : hasLocation$.

The current prototype of Fire handles this case and infers intended result.

4.2.3 Example 3

Consider the expression of a concept *Father*. Using OWL without rules, this can be expressed by the axiom:

$$Father \equiv Male \sqcap \exists hasChild.Child$$

Now consider the expression of a concept *FatherWithKnownChild* to represent a male individual who has an explicitly defined child in the KB. It is more intuitive for non-DL-expert users to express this idea using rules as follows:

$$Male(?x) \wedge hasChild(?x, ?y) \wedge Child(?y) \Rightarrow FatherWithKnownChild(?x)$$

The current prototype of Fire handles such rules and infers intended result.

4.2.4 Example 4

Consider an ontology with the following rules

$$Italian(?x) \Rightarrow WineDrinker(?x)$$
$$French(?x) \Rightarrow WineDrinker(?x)$$

along with the assertion $john : (Italian \sqcup French)$, saying that the individual *john* is an instance of the concept $Italian \sqcup French$. Intuitively, from the open world semantics of OWL, one would expect that since *john* is either *Italian* or *French* (i.e., an instance of at least one of them in any model of the KB), the KB should infer $john : WineDrinker$. But with the reasoning with rules, especially in case of *trigger rules*, this result is not inferred. The reason is that there is no explicit assertion like $john : French$ or $john : Italian$ to trigger either of the above rules. Hence the rules are not applied and the expected result is not inferred.

The current prototype of Fire does not infer the expected result in this case, since it does not yet implement the approach outlined in Section 2.5 to ensure completeness. This kind of inference could also be obtained by treating the rules as logical implications, i.e., by implementing the rules in the contrapositive direction also. Note that the idea behind these two rules can also be expressed in OWL without rules. But the syntax of SWRL-like rules allow users to model such ideas as rules. So this discussion of how the reasoning in Fire handles such cases is relevant here.

4.2.5 Example 5

Consider another ontology with the following rules

$$SmartChild(?x) \Rightarrow Person(?x) \quad (1)$$

$$\neg SmartChild(?x) \Rightarrow Person(?x) \quad (2)$$

along with the assertion $betty : Girl$. According to the open world semantics of OWL both these rules do not fire, since neither $betty : SmartChild$ nor $betty : \neg SmartChild$ is asserted or implied by the KB. But the intention of an application using these rules may *assume* a local closed world assumption for reasoning with these rules. More clearly, since *betty* is not *known* to be an instance of *SmartChild*, she should be a *Person* according to rule (2). This kind of procedural behavior of trigger rules requiring a concept of *local* closed world assumption for reasoning, is addressed in description logics by *epistemic operators*. These operators formalize trigger rules giving it a declarative semantics. This line of work has been thoroughly studied and described in [DLN+92, DLN+98, DLN+94]. In [BKW], the concept of epistemic K-operator introduced in [DLN+98] has been presented specially as an extension to description logics. The basic idea is that rules are reasoned with what

is currently *known* to the KB about its domain, than reasoning with open world knowledge. Interested readers are referred to the mentioned references for details on epistemic operators. With the K-operator applied to the reasoning for this example case (according to [BKW]), the variable in $\neg SmartChild(?x)$ of rule (2) will be bound to all individuals[1] in the KB that are currently *known* to be not instances of the concept *SmartChild*. Hence rule (2) will become applicable for those individuals.

This kind of semantics is not implemented in Fire and hence this example will not result in any rule firing. Moreover, negation of atoms is not allowed in SWRL-like rules. The concept of role negation is not defined in the DL $\mathcal{SHOIN}(D_n)$ and hence a negated role atom has no meaning in OWL DL extended with SWRL-like rules also. Of course, by applying the epistemic semantics to rules as discussed earlier, a local closed world meaning could be defined for a negated role atom.

4.3 Related Work

As a part of the review of literature relevant to the thesis, we present here some of the proposals combining DL ontologies with rules. They can be classified according to the analysis in Section 1.2.

4.3.1 HOOLET

The Hoolet[2] is a naïve approach implementation of an OWL-DL reasoner with support for SWRL rules, from the University of Manchester. The approach followed can be classified under *homogeneous translation based* approach according to Section 1.2. The predicates of concept atoms in the rules are restricted to named classes. Reasoning support is by a straightforward *translation* (based on the semantics of OWL and SWRL) of the ontology into a collection of axioms. These axioms are communicated in the TPTP[3] format to the first order logic (FOL) based theorem prover Vampire (also from the University of Manchester) for consistency checking. The inferences obtained are translated from TPTP format back to OWL.

The approach used for translation is very naïve and not scalable. The FOL reasoning

[1] All individuals *explicitly* defined in the KB, assuming the *active domain semantics*.
[2] http://owl.man.ac.uk/hoolet/
[3] http://www.cs.miami.edu/ tptp/

is undecidable.Since rules and ontology are embedded in a common logic and are not reasoned upon in isolation, cases like the example in Section 4.2.4 yield expected results when modeled in this system.

4.3.2 SWRLJessTab for Protégé

The SWRLJessTab [Gol04] is a Java tab plug-in for for Protégé, bridging Protégé OWL [KMR04], RACER [HMW04] and Jess[4] for reasoning with OWL ontologies extended with SWRL rules. The approach followed can be classified under *homogenous translation based* approach according to Section 1.2. The reasoning approach in the Fire system is quite similar to the approach followed here. DL reasoning is performed on the OWL part using RACER. The entailed facts from OWL are considered for rule inferencing. But unlike Fire's direct DL based rule inferencing, SWRLJessTab translates [OKTM05] entailed facts to Jess facts and SWRL rules to Jess rules. The translated results are communicated to the Jess system using the existing JessTab[5] for Protégé OWL. The execution engine of the Jess system also implements a version of the RETE algorithm [For82] for matching Jess facts with Jess rule patterns, to obtain new rule inferences from the given set of rules. These inferences are translated back to DL and communicated to RACER for obtaining DL entailments that could result from the addition of new inferences to the ontology KB. This is similar to the reasoning approach in Fire.

Although a homogenous translation based approach, the reasoning support in SWRLJessTab is a combination of DL reasoning by RACER and rule reasoning by Jess. Apart from this feature, the rest of the approach is similar in principle to the one followed in Fire. A contradiction introduced by rule inferences become evident only in the end when no rule can possibly fire. It does not yield the expected results for the example case in Section 4.2.4. It uses several available tools and widgets for the entire reasoning process which sometimes becomes cumbersome to use.

4.3.3 SweetRules

The SweetRules[6] project is an integrated set of tools for translating and inferencing (among others) with ontologies and rules. It offers reasoning support for the Description Logic

[4]http://herzberg.ca.sandia.gov/jess/
[5]http://www.ida.liu.se/ her/JessTab/
[6]http://sweetrules.projects.semwebcentral.org

Programs (DLP[7]) subset of OWL and SWRL. Predicates of concept atoms in the rules are restricted to named classes and SWRL built-ins are supported. This can also be classified under *homogenous translation based* approach according to Section 1.2. Based on semantics-preserving[8] the DLP part and the SWRL rules are both translated in multiple steps to Jena2format and then reasoned with the Jena2 reasoning subsystem to obtain inferences. The results are translated back to DLP.

This system is intended as a part of the system that unites several ontology and rule languages to provide services of translating between one another, inferencing, analysis and authoring. It is not clear how this system handles contradictions and entailments that will result in the KB as a consequence of new rule inferences. Its support is limited to only the DLP subset of OWL which is hardly any of OWL DL.

4.3.4 KAON2

KAON2[9] is a an infrastructure implemented in Java, for managing and reasoning with OWL DL ontologies extended with *DL-safe* subset of SWRL rules. The main idea [HM05] behind its implementation is (i) reducing a $\mathcal{SHIQ}$ KB to an equisatisfiable disjunctive Datalog program that entails the same set of ground facts as the original KB, (ii) extending this Datalog program with a finite set of DL-safe rules and (iii) reason with this hybrid logic by reusing the available deductive database optimization techniques like magic sets. The *DL-safe* subset of SWRL is obtained by restricting the predicates of concept atoms in SWRL rules to be named classes and by restricting the rules with the *DL-safety* condition as described in Section 2.4.1. The technique developed in [HMS04] is used to reduce the $\mathcal{SHIQ}$(D) subset of OWL-DL to a disjunctive Datalog program without losing interesting consequences. The reasoning support for the hybrid logic makes use of the deductive database optimization called magic sets. Rules with this DL-safety restriction thus become a *decidable DL-safe* fragment of the SWRL rules.

KAON2 can also be classified under the *homogenous translation based* approach according to Section 1.2. Since both the ontology and rules are translated to axioms in a common logic language for reasoning, cases like the example in Section 4.2.4 yield the expected results.

[7]Description Logic Programs (DLP) is a knowledge representation defined as the expressive intersection of RuleML Logic Programs and OWL/DAML+OIL Description Logic [GHVD03].

[8]http://sweetrules.projects.semwebcentral.org/sweetrules-overview-presentation-2005-04-24-v3.pdf

[9]http://kaon2.semanticweb.org/

4.3.5 AL-Log

The AL-Log [DLNS98] is a knowledge representation system offering reasoning support for the combination of the DL $\mathcal{ALC}$ and the deductive database language Datalog. The Datalog program is *constrained* and *extended* by allowing in the body of the clauses, a set of restrictions on the variables and constants of the clause, expressed in $\mathcal{ALC}$ and called $\mathcal{ALC}$-constraints. Only concept assertion $\mathcal{ALC}$-constraints are allowed, not role assertion constraints. For example, in the following clause defined in the relational subsystem of AL-Log:

$$canTour(X,Y) \ :- \ notSeen(X,Y),\ likesToSee(X,Y)\ \&\ X : Person,\ Y : City$$

the part after '&' are the concept assertion $\mathcal{ALC}$-constraints for this clause. The $X : Person$ constraint on variable X restricts the values of X to range over the set of instances of the specified concept *Person* defined in the structural subsystem of AL-Log. AL-Log offers a sound and complete reasoning support for the hybrid logic, based on constrained SLD-derivation and constrained SLD-resolution. The AL-Log system can be classified under the *hybrid* approach according to Section 1.2.

4.3.6 r-Hybrid KBs

Here we present a recent work also following the *hybrid* approach according to Section 1.2, which seems interesting to explore. The literature [Ros05] provides a general formal framework of r-Hybrid KBs integrating ontologies and rules. It provides a decidable reasoning algorithm for such a framework, provided the logic language (in particular, any DL) $\mathcal{L}$ of the ontology is decidable and is combined with *safe*[10] Datalog$^{\neg\vee}$ program[11]. The results of their work show that OWL DL (the $\mathcal{SHOIN}(D_n)$ description logic) which has decidable reasoning procedures, extended in this framework with safe, positive Datalog rules, preserves the decidability of reasoning. This is extended further to show that reasoning with OWL DL extended in this framework with safe Datalog$^{\neg\vee}$ rules is also decidable [Ros05].

[10]Here *safe* refers to the condition that each variable occurring in the head of a Datalog clause must also occur in the body of the same clause.

[11]A Datalog$^{\neg\vee}$ program allows negation as failure in the body of the rules and disjunction in the head of the rules [Ros05]

Chapter 5

Conclusion

In this chapter we summarize the presented work. Throughout the thesis we presented discussions, relevant existing proposals and details of our approach to addressing the problem of providing a reasoning support for OWL ontologies extended with rules. We proposed the **Fire** system, a rule reasoning engine as a solution to the specific problem of providing direct DL based inferencing service for OWL ontologies extended with a rule language like SWRL, called the SWRL-like rule language. The proposal was supported by a prototype implementation presented. In the following, we summarize the important points that came to light during the process.

- The RETE algorithm implemented for the Fire prototype performs more than 20 times faster than the naïve algorithm for pattern matching (Section 4.1).
- The Fire prototype provides sound, terminating and direct DL based reasoning over SWRL-like rules in contrast to indirect-inference reasoning over rules translated to non-DL rule languages.
- Fire's tight integration with the inference services of the DL reasoner not only makes rule reasoning *in-synch* with OWL reasoning, but also handles (resulting new entailments) and detects (rule-introduced contradictions) the consequences of rule firing.

5.1 Goals Vs. Achievements

Here we summerize the thesis in terms of the goals we set and those we achieved in the thesis. The following *goals* were set by us:

1. Recognize a practically implementable portion of SWRL.
2. Achieve direct DL based reasoning synchronous with OWL reasoning.
3. Offer sound, complete and terminating reasoning.
4. Implement a fast and efficient pattern matcher that scales well with a growing number of assertions and rules.

The following are our *achievements*:

1. We recognized the SWRL-like rule language which differs from SWRL as follows
   - slightly restricts the SWRL syntax for a practical implementation (see Sections 1.4 and 1.5)
   - treats the rules as *trigger rules* in contrast to logical implications in SWRL (see Section 1.2)
   - treats rules with *active domain semantics* to maintain termination of reasoning by restricting application of rules to only explicitly defined individuals in the KB.
2. We proposed the Fire rule reasoning engine and implemented a prototype of Fire to reason directly over DL based SWRL-like rules. The prototype is tightly integrated with the reasoning services of the RACER [HM01] DL reasoner and hence its reasoning is synchronous with OWL reasoning.
3. The prototype of Fire offers *sound* reasoning as it is based on the sound and complete reasoning of RACER over the OWL KB. A rule becomes applicable only if the rule body is satisfied. Facts that match and satisfy patterns in the rule body are essentially answers to nRQL queries of RACER [HMW04]. These answers are guaranteed by RACER to be sound and complete. Thus soundness is inherited by Fire from RACER. *Termination* of Fire is ensured by the active domain semantics restriction (see Section 2.4). We proposed an approach where we conjecture a *complete* reasoning procedure for Fire (see Section 2.5). However, this approach could not be implemented in the prototype of Fire due to the unavailability of a reasoning service from existing DL reasoners. Based on our discussions with the RACER development team, the required service of computing all clash-free completions of an OWL KB

and reasoning over each such completion will be available from RACER in the near future.

4. We implemented the *RETE algorithm* for pattern matching DL rule patterns and DL facts. The implemented RETE algorithm can scale to efficiently handle large amounts of facts (i.e., large ABoxes). It could be possible that if most of the ontology knowledge is defined by a very large number of rules, the RETE network construction could reach a bottle neck in terms of space requirements (see Section 3.2.5 for details on RETE network construction). We implemented an optimization outlined in Section 3.3 to take advantage of similarities existing in rule structures to reduce the space requirements of the RETE network. Another possible similar optimization is identified in Section 5.2.

The reader is referred to Section 4.2 to learn more about what kinds of rules can be handled by the current prototype of Fire. Most examples in that section model typical rules used by the users of ontology development tools like Protégé which allows creation of OWL ontologies and SWRL syntax rules [KMR04].

5.2 Future Work

In the following list we highlight the scope for possible future work to the prototype implementation of Fire.

- *Optimized way of learning 'changes' in the KB*: In the publish-subscribe mechanism provided by RACER [HM01, HMW04], an application can subscribe to receive *changes* in the instance set of a concept. In an event of new instantiations of that concept, RACER publishes only the changes to the listening application. This could be utilized in the rule application component to obtain *updates* that result as a consequence (concept or role assertions) of rule application. We believe this optimization would improve the overall performance of Fire noticeably.

- *Optimizing RETE network construction*: Similar to the implemented optimization described in Section 3.3, instead of two *similar* B-Nodes requiring two JOIN operations that result in identical rows, one *duplicate* JOIN operation could be avoided. This is done by recognizing the structural similarity of the two B-Nodes and making them share the process of obtaining their table rows.

- Include OWL datatype property atoms in the SWRL-like rule language and extend the reasoning support of Fire accordingly.
- Providing Fire as a plug-in for Protégé OWL [KMR04].

Bibliography

[ADG+05] Grigoris Antoniou, Carlos Viegas Damásio, Benjamin Grosof, Ian Horrocks, Michael Kifer, Jan Maluszynski, and Peter F.Patel-Schneider. Combining rules and ontologies. A survey. http://rewerse.net/deliverables/m12/i3-d3.pdf, 2005. REWERSE Deliverables, No. I3-D3, February 2005.

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

[BKW] Franz Baader, Ralf Küsters, and Frank Wolter. Extensions to description logics. In *Description Logic Handbook*, chapter 6, page 232ff.

[BN03] Franz Baader and Werner Nutt. Basic description logics. In Baader et al. [BCM+03], chapter 2, pages 43–95.

[DLN+92] Francesco M. Donini, Maurizio Lenzerini, Daniele Nardi, Andrea Schaerf, and Werner Nutt. Adding epistemic operators to concept languages. In *KR*, pages 342–353, 1992.

[DLN+94] Francesco M. Donini, Maurizio Lenzerini, Daniele Nardi, Werner Nutt, and Andrea Schaerf. Queries, rules and definitions as epistemic sentences in concept languages. In Gerhard Lakemeyer and Bernhard Nebel, editors, *Theoretical Foundations of Knowledge Representation and Reasoning*, number 810 in Lecture Notes in Artificial Intelligence. Springer-Verlag, 1994.

[DLN+98] F. M. Donini, M. Lenzerini, D. Nardi, W. Nutt, and A. Schaerf. An epistemic operator for description logics. *Artificial Intelligence*, 100(1-2):225–274, 1998.

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

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

[GHVD03] Benjamin Grosof, Ian Horrocks, Raphael Volz, and Stefan Decker. Description logic programs: Combining logic programs with description logics. In *Proceedings of the Twelfth International World Wide Web Conference (WWW 2003)*, pages 48–57. ACM, 2003.

[Gol04] Christine Golbreich. Combining rule and ontology reasoners for the semantic web. In *RuleML*, pages 6–22, 2004.

[GR98] Joseph Giarratano and Gary Riley. *Expert Systems: Principles and Programming.*, pages 477–492. The PWS Series in Computer Science. Boston: PWS Pub. Co., third edition, 1998.

[HAMS05] Pascal Hitzler, Jürgen Angele, Boris Motik, and Rudi Studer. Bridging the paradigm gap with rules for OWL. In *W3C Workshop on Rule Languages for Interoperability, 27-28 April 2005, Washington, DC, USA*. W3C, 2005.

[HM01] Volker Haarslev and Ralf Möller. RACER system description. In R. Goré, A. Leitsch, and T. Nipkow, editors, *International Joint Conference on Automated Reasoning (IJCAR 2001)*, pages 701–705, Siena, Italy, June 18-23 2001. Springer-Verlag.

[HM05] Ullrich Hustadt and Boris Motik. Description logics and disjunctive datalog - the story so far. In I. Horrocks, U. Sattler, and F. Wolter, editors, *Proc. International Workshop on Description Logics*, Edinburgh, UK, 2005.

[HMS04] Ullrich Hustadt, Boris Motik, and Ulrike Sattler. Reducing $\mathcal{SHIQ}^-$ description logic to disjunctive datalog programs. In D. Dubois, C. Welty, and M.-A. Williams, editors, *Proceedings of the 9th International Conference on Knowledge Representation and Reasoning (KR2004)*, pages 152–162, Whistler, Canada, June 2004. AAAI Press.

[HMW04] Volker Haarslev, Ralf Möller, and Michael Wessel. Querying the Semantic Web with Racer + nRQL. In *KI-04 Workshop on Applications of Description Logics (ADL 04)*, Ulm, Germany, September 24 2004.

[Hor98] Ian Horrocks. The FaCT system. In H. de Swart, editor, *Principles of Knowledge Representation and Reasoning: Proceedings of the Sixth International Conference (KR'98)*, number 1397 in Lecture Notes in Artificial Intelligence, pages 307–312. Springer-Verlag, May 1998.

[HPB+04] Ian Horrocks, Peter F. Patel-Schneider, Harold Boley, et al. SWRL: A semantic web rule language combining OWL and RuleML. http://www.w3.org/Submission/2004/SUBM-SWRL-20040521/, May 21 2004. W3C Member Submission 21 May 2004.

[HPS04] Ian Horrocks and Peter F. Patel-Schneider. A proposal for an OWL rules language. In *The Thirteenth International World Wide Web Conference*, New York, New York, May 2004. ACM Press.

[KMR04] Holger Knublauch, Mark A. Musen, and Alan L. Rector. Editing description logic ontologies with the protégé OWL plugin. In *International Workshop on Description Logics*, Whistler, BC, Canada, 2004.

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

[MSS04] Boris Motik, Ulrike Sattler, and Rudi Studer. Query answering for OWL-DL with rules. In Sheila A. McIlraith, Dimitris Plexousakis, and Frank van Harmelen, editors, *The Semantic Web ISWC 2004: Third International Semantic Web Conference, Hiroshima, Japan, November 7-11, 2004. Proceedings*, Lecture Notes in Computer Science, page 549. Springer-Verlag GmbH, 2004.

[OKTM05] Martin O'Connor, Holger Knublauch, Samson Tu, and Mark Musen. Writing rules for the semantic web using SWRL and Jess. http://www.med.univ-rennes1.fr/~cgolb/Protege2005/SWRLJessOConnor.pdf, 2005. 8th Intl. Protégé Conference, July 18-21, 2005, Madrid, Spain.

[OWL04] OWL web ontology language overview. http://www.w3.org/TR/owl-features/, 2004. W3C Recommendation 10 February 2004.

[Rec02] Alan Rector. Analysis of propagation along transitive roles: Formalisation of the GALEN experience with medical ontologies. In *In Proceedings of DL 2002*, Toulouse, France, 2002.

[Ros05] Riccardo Rosati. On the decidability and complexity of integrating ontologies and rules. *Journal of Web Semantics*, 2005. To appear.

Appendix A

Trace of an Example

We present here a partial trace of executing the presented prototype of **Fire** with an example SWRL-like ontology file. First we give the details of the example ontology and then describe the program trace when executed with the current prototype of Fire. Details of the tools used are as follows: (i) the RacerPro 1.8.0 system as the DL reasoner, (ii) the Protégé SWRL Factory (from OWL Plugin 2.0 beta for Windows) for parsing the SWRL-like rules from the ontology file, (iii) the JRacer Java API for communication with the reasoner and (iv) a machine with AMD Athlon 1.86 GHz processor, 512 MB of RAM and the Windows XP operating system.

A.1 The Example Ontology – family.owl

The example ontology *family.owl* was designed as a part of the present work. It represents a family ontology with the usual relationships. An overall glimpse of the *family.owl* ontology is shown in Figure 6. Specifically, the pictorial depiction shows the initially asserted facts in the ontology knowledge base (KB). The facts initially asserted are shown in Table 2. The SWRL-rules defined in the KB are listed below:

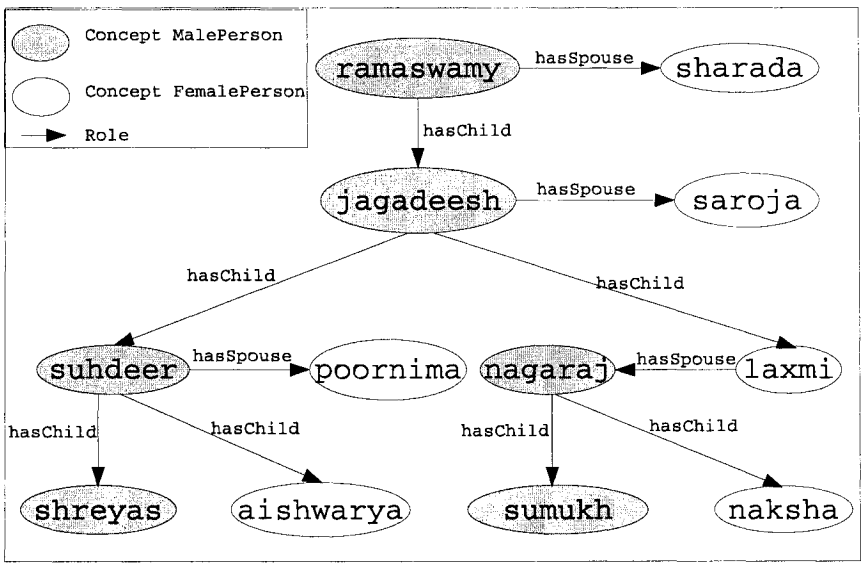


Figure 6: Pictorial illustration of the initially asserted facts in *family.owl* ontology.

$$hasParent(??x, ??y) \wedge FemalePerson(?y) \Rightarrow hasMother(?x, ?y) \quad (3)$$

$$hasChild(?x, ?y) \wedge MalePerson(?y) \Rightarrow hasSon(?x, ?y) \quad (4)$$

$$hasParent(?x, ?y) \wedge hasSister(?y, ?z) \Rightarrow hasAunt(?x, ?z) \quad (5)$$

$$hasChild(?x, ?y) \wedge hasSpouse(?x, ?z) \Rightarrow hasChild(?z, ?y) \quad (6)$$

$$hasParent(?x, ?y) \wedge hasChild(?y, ?z) \Rightarrow Sibling(?x) \quad (7)$$

$$hasSibling(?x, ?y) \wedge FemalePerson(?y) \Rightarrow hasSister(?x, ?y) \quad (8)$$

$$hasChild(?x, ?y) \wedge FemalePerson(?y) \Rightarrow hasDaughter(?x, ?y) \quad (9)$$

$$hasParent(?x, ?y) \wedge hasChild(?y, ?z) \Rightarrow hasSibling(?x, ?z) \quad (10)$$

$$hasParent(?x, ?y) \wedge hasBrother(?y, ?z) \Rightarrow hasUncle(?x, ?z) \quad (11)$$

$$hasSibling(?x, ?y) \wedge MalePerson(?y) \Rightarrow hasBrother(?x, ?y) \quad (12)$$

$$hasParent(?x, ?y) \wedge MalePerson(?y) \Rightarrow hasFather(?x, ?y) \quad (13)$$

In this example the SWRL-like rules make it possible to define certain relationships which cannot be expressed in OWL without rules, e.g., the rules 5, 6, 10 and 11.

Concept Membership	
FemalePerson:	{sharada, saroja, poornima, laxmi, aishwarya, naksha}
MalePerson:	{ramaswamy, jagadeesh, sudheer, nagaraj, shreyas, sumukh}
Role Membership	
hasSpouse:	{(ramaswamy,sharada), (jagadeesh,saroja), (sudheer,poornima), (laxmi,nagaraj)}
hasChild:	{(ramaswamy,jagadeesh), (jagadeesh,sudheer), (jagadeesh,laxmi), (sudheer,shreyas), (sudheer,aishwarya), (nagaraj,sumukh), (nagaraj,naksha)}

Table 2: Initially asserted facts in the *family.owl* ontology.

A.2 Trace of family.owl

Here we present selected excerpts from the trace of executing *family.owl* with Fire, with explanations inserted in-between where necessary. The execution starts as follows:

```
MODE 'DEFAULT'
READING FILE: file:/e:/family.owl
PROTEGE SWRL API READING RULES FROM FILE: file:/e:/family.owl
RULES READ
1 hasParent(x,y) FemalePerson(y) --> hasMother(x,y)
2 hasChild(x,y) MalePerson(y) --> hasSon(x,y)
...
11 hasParent(x,y) MalePerson(y) --> hasFather(x,y)
```

The *default* mode indicates that Fire is not running in the debugging mode (see Section 3.1.4). The supported rules parsed by the parser are listed. The functioning of the pattern matcher follows:

```
RETE NETWORK CONSTRUCTED
RACER READING ONTOLOGY FILE: file://e:/family.owl
ABOX REALIZED
ASSERTIONS READ
```

```
1 MalePerson(shreyas)
2 MalePerson(sumukh)
...
17 hasParent(sumukh,nagaraj)
...
24 hasChild(nagaraj,sumukh)
...
31 hasSpouse(saroja,jagadeesh)
32 hasSpouse(nagaraj,laxmi)
33 hasSpouse(laxmi,nagaraj)
34 hasSpouse(jagadeesh,saroja)
```

The RETE network is constructed by referring to the structure of the parsed rules. For each parsed rule, the network has a T-Node. So in this example, eleven T-Nodes named $T1$ through $T11$ are created, with node $T1$ corresponding to the first rule listed, $T2$ corresponding to the second rule listed, and so on. The pattern matcher component of Fire obtains the facts (asserted and entailed concept and role assertions) in the KB from the RACER reasoner using nRQL queries. The list showing totally 34 of these facts indicate that apart from the 23 initially asserted facts (see Figure 6 and Table 2) several entailed facts were discovered by the DL reasoner. These are included for the pattern matching process to identify which rules become applicable. The execution is transferred to the rule application component along with the instantiations of the applicable rules. The following partial trace refers to the outcome of the first *reasoning cycle*.

```
T-NODE: T2
NEW INFERENCES
1 hasSon(sudheer,shreyas)
2 hasSon(ramaswamy,jagadeesh)
3 hasSon(nagaraj,sumukh)
4 hasSon(jagadeesh,sudheer)
NO NEW INFERENCES IN THIS NODE...NEXT NODE.
T-NODE: T4
NEW INFERENCES
```

```
1 hasChild(poornima,aishwarya)
2 hasChild(poornima,shreyas)
3 hasChild(sharada,jagadeesh)
4 hasChild(laxmi,naksha)
5 hasChild(laxmi,sumukh)
6 hasChild(saroja,sudheer)
7 hasChild(saroja,laxmi)
...
```

Only the two rules represented by nodes $T2$ and $T4$ were triggered with 4 and 7 *new* instantiations respectively, in the first reasoning cycle. Note that the message `NO NEW INFERENCES IN THIS NODE...NEXT NODE` means that the intermediate node $T3$ did not yield any *new* inferences. The new inferences identified were found to be consistent with the KB and were *materialized* in the ABox of the reasoner in a batch. They will be included for identifying rule instantiations in the next reasoning cycle. The outcome of the next cycle follows:

```
T-NODE: T2
NEW INFERENCES
1 hasSon(poornima,shreyas)
2 hasSon(sharada,jagadeesh)
3 hasSon(laxmi,sumukh)
4 hasSon(saroja,sudheer)
NO NEW INFERENCES IN THIS NODE...NEXT NODE.
NO NEW INFERENCES IN THIS NODE...NEXT NODE.
T-NODE: T5
NEW INFERENCES
1 Sibling(aishwarya)
2 Sibling(shreyas)
3 Sibling(sudheer)
4 Sibling(naksha)
5 Sibling(sumukh)
6 Sibling(laxmi)
```

```
NO NEW INFERENCES IN THIS NODE...NEXT NODE.
T-NODE: T7
NEW INFERENCES
1 hasDaughter(sudheer,aishwarya)
2 hasDaughter(nagaraj,naksha)
3 hasDaughter(jagadeesh,laxmi)
4 hasDaughter(poornima,aishwarya)
5 hasDaughter(laxmi,naksha)
6 hasDaughter(saroja,laxmi)
T-NODE: T8
NEW INFERENCES
1 hasSibling(aishwarya,shreyas)
2 hasSibling(shreyas,aishwarya)
3 hasSibling(sudheer,laxmi)
4 hasSibling(naksha,sumukh)
5 hasSibling(sumukh,naksha)
6 hasSibling(laxmi,sudheer)
```

Consider for instance, the new inference `2 hasChild(poornima,shreyas)` from node $T4$, materialized in the previous cycle. This triggered the `hasSon` rule which yields the new inference `1 hasSon(poornima,shreyas)` from node $T2$ in this cycle. The subsequent reasoning cycles continue in this way, until Fire is driven to a termination either by a contradiction introduced by a rule inference or by not inferring any *new* rule inference. These examples highlight the importance of Fire's tight integration with DL inferencing and shows that Fire is *in-synch* with RACER's inferences.

The same ontology was added with another simple rule

$$FemalePerson(?x) \Rightarrow MalePerson(?x)$$

to introduce a contradiction. In the following execution trace, we call this altered ontology as *familyc.owl* and this rule corresponds to the T-Node $T9$.

```
MODE 'DEFAULT'
READING FILE: File:/e:/familyc.owl
PROTEGE SWRL API READING RULES FROM FILE: file:/e:/familyc.owl
```

```
RULES READ
1 hasParent(x,y) FemalePerson(y) --> hasMother(x,y)
2 hasChild(x,y) MalePerson(y) --> hasSon(x,y)
3 hasParent(x,y) hasSister(y,z) --> hasAunt(x,z)
4 hasChild(x,y) hasSpouse(x,z) --> hasChild(z,y)
...
9 FemalePerson(x) --> MalePerson(x)
...
RETE NETWORK CONSTRUCTED
RACER READING ONTOLOGY FILE: file://e:/familyc.owl
ABOX REALIZED
ASSERTIONS READ
1 MalePerson(sumukh)
...
34 hasSpouse(jagadeesh,saroja)
```

The trace for the altered file *familyc.owl* begins in the same way. Notice the rule 9 corresponding to the node $T9$ as mentioned earlier. The pattern matcher component identifies the applicable rules and their corresponding sets of rule instantiations. Among them the *new* ones identified and processed by the rule application component are shown in the following:

```
T-NODE: T2
NEW INFERENCES
1 hasSon(sudheer,shreyas)
2 hasSon(ramaswamy,jagadeesh)
3 hasSon(nagaraj,sumukh)
4 hasSon(jagadeesh,sudheer)
5 hasSon(sudheer,aishwarya)
6 hasSon(nagaraj,naksha)
7 hasSon(jagadeesh,laxmi)
NO NEW INFERENCES IN THIS NODE...NEXT NODE.
T-NODE: T4
```

```
NEW INFERENCES
1 hasChild(poornima,aishwarya)
2 hasChild(poornima,shreyas)
3 hasChild(sharada,jagadeesh)
4 hasChild(laxmi,naksha)
5 hasChild(laxmi,sumukh)
6 hasChild(saroja,sudheer)
7 hasChild(saroja,laxmi)
NO NEW INFERENCES IN THIS NODE...NEXT NODE.
NO NEW INFERENCES IN THIS NODE...NEXT NODE.
...
T-NODE: T9
NEW INFERENCES
1 MalePerson(aishwarya)
2 MalePerson(laxmi)
3 MalePerson(naksha)
4 MalePerson(poornima)
5 MalePerson(saroja)
6 MalePerson(sharada)
...

CHECKING CONSISTENCY
CONSISTENCY RESPONSE -> NIL ABOX IS INCONSISTENT. QUITTING
```

The new inferences in nodes $T2$ and $T4$ are the same as obtained in the trace of *family.owl*. But the outcome of the node $T9$ indicates that the individuals that were asserted initially as instances of the concept *FemalePerson* are to be inferred as instances of the concept *MalePerson*.[1] The rule application component discovers that these new inferences from the node $T9$ of this cycle contradict the KB. Hence they are not materialized and Fire is driven to a termination. In such a case, the user or the application could run Fire in the *debugging mode* (see Section 3.1.4) to discover a specific rule instantiation that is causing a contradicting rule inference. This would help in identifying a possible unintended error

[1]Here the concepts *MalePerson* and *FemalePerson* are assumed to be defined as disjoint in the ontology, as per usual understanding of family relationships.

in the rule base. Since a contradiction was identified in the first reasoning cycle itself, no materialized inferences from previous cycles are returned to the user or the application.