

A Novel SWRL-based ABox Reasoner for SWRL-enabled OWL Ontology

Qingmai Wang and Xinghuo Yu

RMIT University,
Melbourne
Victoria, Australia
qingmai.w@gmail.com, x.yu@rmit.edu.au

Abstract. Semantic Web Rule Language (SWRL) is an effective complement to Web Ontology Language (OWL) for building ontologies. It allows users to write Horn-like rules that can be expressed in terms of OWL classes and that can reason about OWL individuals. The support of SWRL in most of the existing ontology reasoners is done by translating both OWL and SWRL into some third party reasoning engines such as Jess, Jena or Sesame, which is not efficient. In this paper, a novel SWRL-based reasoner which directly works on SWRL rules and supports the translation from OWL-DL to SWRL is proposed, and is applied to ABox reasoning of the OWL-DL ontology.

1 Introduction

Web Ontology Language (OWL) has now been accepted as an ideal language for constructing ontologies for the emerging semantic web technologies. OWL is a family of knowledge representation languages which is layered on the Resource Description Framework (RDF) [12]. The family includes three sub-languages with different levels of expressivity: OWL-Lite, OWL-DL and OWL-Full. This paper focuses on OWL-DL which is essentially Description Logics (DL) [5]. The language contains rich human readable syntax and terms to represent domain concepts and relationships between those concepts accurately and consistently.

OWL-DL ontology, as a DL based ontology, doesn't support rules, which limits expressivity. The importance of extending DL based ontology to support rules has been widely realized and some works have already been done to address this issue, e.g. AL-Log [4], CARIN [11] and most importantly, Semantic Web Rule Language (SWRL) [6][7].

SWRL is a rule description language which combines Datalog and OWL. The language allows users to write Horn-like rules that can be expressed in terms of OWL classes and that can reason about OWL individuals [7]. More and more OWL ontologies are developed with SWRL rules, the reasoning of SWRL enabled OWL ontologies are thus becoming important. Most of the existing systems address this issue by translating both OWL and SWRL clauses to some third party inference engines such as Jess, Jena or Sesame [13] [15]. In addition, the conventional tableau-based reasoning algorithm is not efficient with large

ABoxes. It requires all the knowledge to perform ABox reasoning such as query, which is not necessary. It has been proved that using rule-based reasoning is more efficient with large ABoxes, e.g. Datalog rules [14]. But none of them use SWRL directly.

This paper proposes a novel SWRL-based reasoner which directly works on SWRL rules and supports the translation from OWL-DL to SWRL. The reasoner firstly translates OWL-DL clauses to SWRL rules to unify the format of rules. The syntax SWRL is extended for stronger expressivity making it more compatible with OWL-DL. After that, the reasoner consequently performs backward reasoning on the unified rule base of the ontology. A heuristic search engine, which is applied to search for solutions for a fired rule, is developed to optimize the performance.

The rest of the paper is organized as follows: section 2 introduces the backgrounds of the reasoner; details of the reasoner is described in section 3; section 4 explains the reasoning algorithm with a simple example and section 5 concludes the paper.

2 Backgrounds

This section briefly introduces background knowledge of this paper. The definition and semantics of OWL-DL and SWRL are firstly discussed. The undecidability issues about SWRL, which are important for the following discussion regarding the termination of the reasoning process in Section 3, are then introduced

2.1 OWL&SWRL

OWL is defined as an extension to RDF in the form of a vocabulary entailment, so that the syntax of OWL is the syntax of RDF and the semantics of OWL are an extension of the semantics of RDF [8]. Since both OWL and RDF are encoded in XML documents, the concrete syntax of OWL is the parse of XML codes.

Fortunately, OWL-DL, a subset of OWL, mostly follows the description logic $SHOIN(\mathcal{D})$. So an abstract syntax and semantics of OWL-DL constructs can be demonstrated with their DL mappings, and is described in Tab.1 [16] (C, D denote classes).

SWRL is a combination of the OWL-DL with the Unary/Binary Datalog RuleML language [7]. The language allows user to write Horn-like rules that can be expressed in terms of OWL constructs. A rule takes the form of implication between an antecedent (body) and a consequent (head). A rule body may contain multiple atoms, and is treated as a conjunction of these atoms. The intended meaning of a rule can be read as: whenever the atoms specified in the antecedent hold, the atom in the consequent must also hold. Forms of SWRL atom can be found in Tab.2 as well as their semantics [3].

Table 1. OWL-DL syntax and semantics

Construct Name	OWL Syntax	DL syntax	DL Semantics
Atomic Class	$A(\text{URI})$	A	$A^I \subseteq \Delta^I$
Universal Class	owl:Thing	\top	$\top^I = \Delta^I$
Object Property	$R(\text{URI})$	R	$R^I \subseteq \Delta^I \times \Delta^I$
Datatype Property	$U(\text{URI})$	U	$U^I \subseteq \Delta^I \times \Delta^D$
Conjunction	$\text{intersectionOf}(C, D)$	$C \sqcap D$	$(C \sqcap D)^I \subseteq C^I \cap D^I$
Disjunction	$\text{unionOf}(C, D)$	$C \sqcup D$	$(C \sqcup D)^I \subseteq C^I \cup D^I$
Negation	$\text{ComplementOf}(C)$	$\neg C$	$(\neg C)^I = \Delta^I / C^I$
someValue Restr.	$\text{restriction}(R \text{ someValuesFrom}(C))$	$\exists R. C$	$(\exists R. C)^I = \{x \in \Delta^I \mid \exists y \in \Delta^I, (x, y) \in R^I, y \in C^I\}$
allValues Restr.	$\text{restriction}(R \text{ allValuesFrom}(C))$	$\forall R. C$	$(\forall R. C)^I = \{x \in \Delta^I \mid \forall y \in \Delta^I, (x, y) \in R^I \rightarrow y \in C^I\}$
Cardinality Restriction	$\text{restriction}(R \text{ minCardinality}(n))$	$\geq nR$	$(\geq nR)^I = \{x \in \Delta^I, \ \ y, (x, y) \in R^I\ \geq n\}$
	$\text{restriction}(R \text{ maxCardinality}(n))$	$\leq nR$	$(\leq nR)^I = \{x \in \Delta^I, \ \ y, (x, y) \in R^I\ \leq n\}$

Table 2. Forms and Semantics of SWRL atoms

$C(?x/I)$	C is an OWL Class, $?x$ is a variable, I is an OWL individual
$P(?x/I, ?y/I/D)$	P is an OWL Property, $?x, ?y$ are variables, I is an OWL individual, D is an OWL data value
$\text{sameAS}(?x/I, ?y/I)$	$?x, ?y$ are variables, I is an OWL individual
$\text{differentFrom}(?x/I, ?y/I)$	
$\text{builtIn}(r, D_1, D_2 \dots)$	r is a built-in relation, $D_1, D_2 \dots$ are OWL data values.

2.2 Issues on undecidability

OWL-DL is decidable due to that $\mathcal{SHOIN}(\mathcal{D})$ can express only local properties [2], while Datalog is also decidable since it is mostly applied to finite data model, namely the reasoning of Datalog is under Closed World Assumption (CWA). SWRL, a combination of OWL-DL and Datalog, however becomes undecidable for two reasons: adopts the Open World Assumption (OWA) which makes the data model infinite; the rule syntax can represent property chain which allows an individual to be connected with other individuals that are arbitrarily far away.

The undecidability of SWRL has been realized since it was developed. This means it is not possible to design an effective reasoning algorithm for SWRL rules which guarantees that decision can be made in a finite time. The proof of the undecidability can be shown by encoding a known undecidable domino problem with SWRL [6]. Several approaches have been proposed to address this problem by limiting the expressivity, e.g. DL-safe Rules [10].

In this paper, to ensure that the reasoning can always be completed in finite time, the SWRL-based reasoner is processed under CWA. This design is similar to the proposed DL-safe rules which bind all variables only to known individuals in ontology so that unknown individuals are interpreted negatively (equivalent to the meaning of CWA). Adopting OWA or CWA mainly depends on the application. Both OWL and SWRL are developed as OWA-based languages because originally they focus on Semantic Web area in which the internet can be seen as an unlimited knowledge resource. But since the application of SWRL enabled OWL ontology has been extended to some areas which adopts CWA, e.g. Com-

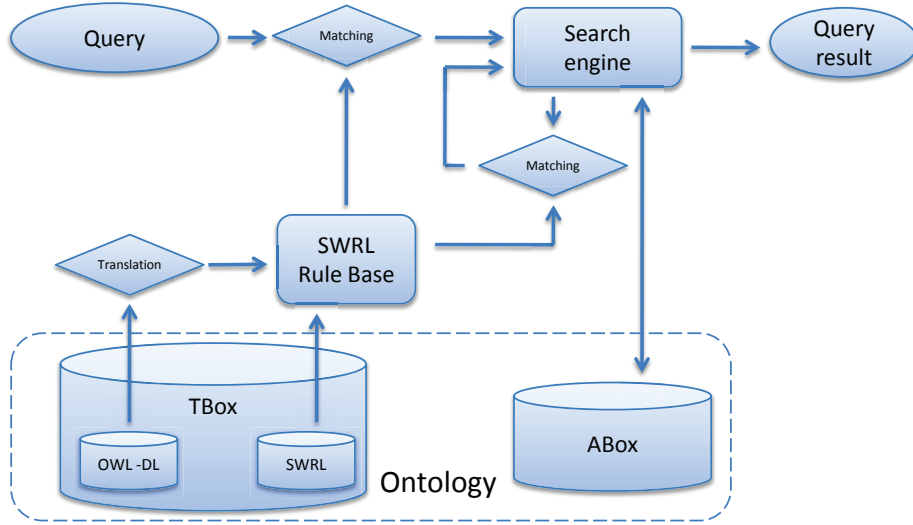


Fig. 1. Framework of SWRL-based reasoner

puter Aided Design (CAD) [9] [1], a reasoner which supports the reasoning for the combination of OWL and SWRL under CWA is necessary.

3 SWRL-based Reasoner

In this section, the details for the proposed reasoner is discussed. The reasoner is designed for ABox reasoning and is demonstrated using Instance Retrieval (IR) task: to retrieve instances that belongs to a certain logic expression. IR is also called query, and is an important component of ABox reasoning. Fig.1 shows the framework of the reasoner. The system firstly translates some of the OWL-DL clauses in the TBox to SWRL rules to unify the format of rules. A unified SWRL rule base is thus created plus those original SWRL rules in the TBox. Once a query is inputted, the rule which matches the query is fired. A heuristic search engine is then applied to the fired rule to search for solutions which bind individuals to rule variables so that the atoms in the rule body holds true. The search engine requires information from the ABox and also may fire some other rules if the required information need further inference. The whole reasoning process is essentially a backward chaining. After the searching is finished, the query results is then generated based on the found solutions.

Translating OWL-DL to SWRL is complicated. This is because SWRL and OWL-DL have different levels of expressivity. They both have constructs that the other cannot express. The syntax of SWRL is extended in this paper so that most of the constructs of OWL-DL can be translated to SWRL.

The rest of this section is organized as follows: The basic translation from OWL-DL to SWRL without any extension is discussed firstly; the process of

backward reasoning are showed secondly; details of the search engine are then described; finally the extension of SWRL is discussed as well as its impact on the reasoning process.

3.1 Translating OWL-DL to SWRL

The translation from OWL-DL to SWRL without any extension is the process of exploring the overlap of expressivity between OWL-DL and SWRL. In this case, SWRL is seen as a syntax sugar of OWL-DL [3] since SWRL rules are more readable than OWL-DL clauses. The mapping of equivalent constructs between OWL-DL and SWRL is shown in Tab.3

Table 3. Mapping of equivalent constructs between OWL-DL and SWRL

OWL-DL	SWRL
<i>Class: A, B</i>	$A(?x), B(?x)$
<i>Property: R</i>	$R(?x, ?y)$
$A \sqcap B$	$A(?x) \wedge B(?x)$
$\exists R.A$	$R(?x, ?y) \wedge A(?y)$

Consider an OWL-DL equivalent axiom as an example (shown in $\mathcal{SHOIN}(\mathcal{D})$ format):

$$Research_Student \equiv Student \sqcap \exists publish.Paper$$

The axiom defines a class “*Research_Student*” using its Necessary and Sufficient (N&S) condition “*Student* \sqcap $\exists publish.Paper$ ”, where “*Student*, *Paper*” are Atomic Classes and “*publish*” is an Object Property. The intended meaning is “a student who has published some papers is a research student”. With the mapping shown in Tab.3, a SWRL rule can be created as:

$$Student(?x) \wedge publish(?x, ?y) \wedge Paper(?y) \rightarrow Research_Student(?x)$$

3.2 Reasoning Process

Adopting backward chaining, the reasoner ensures that only the knowledge regarding the task goal is required. This is much more efficient than Tableau algorithm while dealing with large ABoxes since Tableau requires all the ABox knowledge to get results. It should be noticed that while using backward chaining the antecedent of a rule must necessarily and sufficiently imply the consequent. This assumption must be imposed to SWRL rules otherwise the system cannot decide whether the consequent is true or false if the antecedent cannot be proved to be true.

The reasoning process of the reasoner is demonstrated by executing a query task. Let E denote the expression of a query. The expression of query in this paper is limited to three forms:

1. $C(?)$:retrieve instances that belongs to the class C ;
2. $P(? , a)$:retrieve instances whose value on property P is a ;
3. $P(a, ?)$:retrieve instances which is the value of a on property P ;

Let C, P, R be an arbitrary class, property and rule respectively; $\mathbf{T}, \mathbf{A}, \mathbf{R}$ denote TBox, ABox and rule base respectively; \mathcal{U} denotes the collection of retrieved instances, and is initially set to empty. Once a query expression E is determined, the reasoning process can be described as follows:

1. Asserted knowledge checking: Checking the asserted knowledge of ontology, if it has already been asserted in \mathbf{A} that some individuals (the collection of which is written as \mathbf{I}) which are the instances of E , then

$$\mathcal{U} = \mathcal{U} \cup \mathbf{I}$$

2. Rule matching: Searching the rule base, for each rule in \mathbf{R} , if there exists a rule R whose consequent has the same predicate as E , R is fired. Otherwise if no rule can be fired, the reasoning process is terminated and returns \mathcal{U} .
3. Searching for solutions for the fired rule: After a rule is fired, the system searches for solutions for the fired rule. In a solution, all the variables in the rule is bound to a value (value can be individuals or datatype values such as integer and string), which makes all the atoms in the rule antecedent hold true. There may be more than one solutions for a rule. For each solution, assuming query E refers to the variable v (the position of v in the atom of the fired rule consequent is the same as the position of $?$ in the query E) and v is bound to a value a in the solution, then

$$\mathcal{U} = \mathcal{U} \cup \{a\}, \text{ if } a \notin \mathcal{U}$$

4. Sub-query: To bind a value to each variable, the above searching step requires to retrieve instances for each atom in the antecedent of the fired rule (shown in the example in section 4). A sub-query for each atom is then started. The sub-query may fire another rule and re-do the search algorithm for new fired rules until no more rule can be fired. This is essentially a backward reasoning which backward tracks rules from the task goal to the ABox knowledge.
5. Returning results: After the reasoning process is completed, return \mathcal{U} as the query results for E .

3.3 Search engine

The search engine searches for solutions for a fired rule, which is complicated. Let's consider an exhaustive search. Assuming that there are n variables, and each of them has m possible values, the system will generate n^m combinations of values for all variables. For each combination, the system needs to check whether it can make the antecedent of a fired rule true or false. If it returns true, the combination is returned as a solution. This method may take extremely long time to find a solution in large ABoxes since there may be thousands of possible

values for a variable. To address this issue, a heuristic search algorithm is then developed.

In the heuristic search algorithm, a search tree is constructed. The root of the tree is a state that values of all the variables are unknown. The system then selects a variable and binds the variable to each value in its value range (value range is a collection of all the possible values of a variable), so that branches are generated for each value. It is important for the system to select the variable which have minimal size of value range since minimal number of branches will be generated in this case. In each branch, for that one of the variables has got a value, the size of the value range of other variables which are related to the previous variable will be greatly reduced. Redoing the above process for other variables and generate more sub-branches until all variables have got a value, a solution is then generated.

To describe the searching algorithms in detail, some terms and notations are defined as follows: atom is written as A ; the collection of comparison atoms (The atoms which compare individuals or data, e.g., `sameAS`, `differentFrom`) in the antecedent is written as **CA**s; the collection of atoms other than **CA**s is written as **RA**s; variable and solution are written as v, s ; vl denotes a value; value range of a variable is written as **VR**; **V, S** denote the collection of all the variables in the rule antecedent and the collection of all the returned solutions respectively; binding a variable v to a value vl can be written as $v \leftarrow vl$.

The searching algorithm is then described as follows:

1. Pre-processing: By analyzing the predicates of all the atoms in **RA**s, for each v in **V**, from TBox **T** it is easy to locate a class C that refers to v (for unary atoms, the predicate of the atom indicates the class, and for binary atoms, the class information can be found through the domain and range of the relevant property). If there aren't any rules or axioms which imply C , set **VR** of v to the collection of members of C (asserted in the ABox), otherwise set **VR** to \top .
2. Searching: After all the variables have been initially arranged a value range, the system starts searching. The algorithm is shown in the pseudo-code in Algorithm 1:
3. Termination: The termination of the search algorithm is ensured by adopting CWA. As is shown in function $Query(A, v)$, since there aren't any unknown or unnamed individuals, the empty value range of any variable indicates that the variable is unsolvable, so that the function returns false. If the function returns false for all the possible values of v , the search algorithm is then terminated and the system can't find any solution for the antecedent of the rule.
4. Completion: Once the searching is completed, a collection of solutions **S** is returned.

3.4 Extending SWRL for stronger expressivity

Two kinds of rules which semantically mean "implication" have been implemented in the SWRL enabled OWL-DL ontology. One is explicitly described in

Algorithm 1 Heuristic Search Algorithm

```

function SEARCH( $s$ ,  $\mathbf{RAs}$ ,  $\mathbf{V}$ )
  if there are at least one variable in  $\mathbf{V}$  whose value is unknown then
    select a  $v$  from  $\mathbf{V}$  if the value of  $v$  is unknown and
    the  $v$  has minimal size of  $\mathbf{VR}$  in all the variables in  $\mathbf{V}$ 
     $\triangleright$  ensures that the search tree have minimal number of branches
  if  $\mathbf{RAs}$  is not empty then
    for all  $vl \in \mathbf{VR}$  of  $v$  do
      set  $v \leftarrow vl$ 
      for all  $A \in \mathbf{RAs}$  and  $A$  contains  $v$  do
        if Query( $A, v$ ) returns false then
          claims an error
        else
          remove  $A$  from  $\mathbf{RAs}$ 
        end if
      end for
      if there is no error then
        add  $v \leftarrow vl$  to  $s$ 
        Search( $s$ ,  $\mathbf{RAs}$ ,  $\mathbf{V}$ )
      end if
    end for
  end if
else  $\triangleright$  all the variables have been bound to a value
  for all  $A \in \mathbf{CAs}$  do
    if comparison in  $A$  returns false with bound values then
      claims an error
    end if
  end for
  if there is no error then
    add  $s$  to  $\mathbf{S}$ 
  end if
end if
end function

function QUERY( $A, v$ )
  if  $A$  is unary atom then
    assuming  $A$  is in the form of  $C(?x)$ , execute query  $C(?)$ 
    if value of  $v$  is a member of query result then
      return true
    else
      return false
    end if
  else if  $A$  is Binary atom then
    if  $A$  is in the form of  $P(?x, ?y)$  where  $v$  is  $?x$  then
      assuming the value of  $?x$  is  $vl$ , execute query  $P(vl, ?)$ , then update
      the  $\mathbf{VR}$  of  $?y$  to be the intersection of old  $\mathbf{VR}$  and query result
      if the intersection is not empty then
        returns true
      else
        returns false
      end if
    else if  $A$  is in the form of  $P(?x, ?y)$  where  $v$  is  $?y$  then
      assuming the value of  $?y$  is  $vl$ , execute query  $P(? , vl)$ , then update
      the  $\mathbf{VR}$  of  $?x$  to be the intersection of old  $\mathbf{VR}$  and query result
      if the intersection is not empty then
        returns true
      else
        returns false
      end if
    end if
  end if
end function

```

SWRL; the other is implicitly described by the equivalent classes axiom which use N&S condition to define classes in OWL-DL. To effectively reason through the whole ontology, as is discussed above, there is a need to unify the rule formats, which means translating OWL-DL equivalent classes axiom to SWRL rules. However, since these two languages have different levels of expressivity, there are lots OWL-DL constructs that don't have any corresponding terms in SWRL. In this section, the syntax of SWRL is extended to cover more OWL-DL constructs. The impacts of these extensions on the reasoning algorithm are also discussed.

Universal Quantifier: In a SWRL rule, all the variables are universally quantified over the rule. Noticing that we have:

$$\forall x(P(?x) \rightarrow Q) \implies (\exists ?x P(?x)) \rightarrow Q$$

Existential quantifier is thus enforced to all the variables in the antecedent of the rule. So it is difficult to represent *allValuesFrom* restriction as below:

$$C \equiv A \sqcap \forall R.B$$

Let's extend SWRL by introducing the following mapping:

$$\forall R.A \iff R(?x, ?y) \wedge (\forall ?y)A(?y)$$

So that the above example can be translated as:

$$A(?x) \wedge R(?x, ?y) \wedge (\forall ?y)B(?y) \rightarrow C(?x)$$

While applying the search algorithm to the rule with universal quantifier, the system firstly remove all the atoms with universal quantifier (e.g. $B(?y)$ in the above example) from **RA**s. The collection of removed atoms is written as *UAs*. Then the system search for the solutions to find all the possible values of the universally quantified variables (*Uvs*) without the consideration of *UAs*. Finally the system verifies all the solutions using *UAs*. If all the *UAs* returns true for all the possible values of *Uvs* in all the solutions, the search algorithm is then finished and returns all the solutions as the result, otherwise all the solutions are negated and the search fails.

Cardinality restriction: The mapping of the cardinality restriction to SWRL is shown in tab.4

Considering an SWRL rule which contains an atom " $\geq n ?x R(?x, ?y)$ ", in the searching process, the system firstly searches for solutions as normal. Once a list of solutions is found, if there are no less than n solutions in which the values of variable $?x$ are same and the values of variable $?y$ are different, the solutions are then verified, otherwise all the solutions are negated.

Table 4. Mapping of cardinality restriction from OWL-DL to SWRL

OWL-DL	SWRL
$\geq n R$	$\geq n ?x R(?x, ?y)$
$\leq n R$	$\leq n ?x R(?x, ?y)$

Negation: The mapping of the negation is shown as:

$$\neg C \implies \text{not}C(?x)$$

Considering a unary atom in SWRL is in the negation form, the $\text{Query}(A, v)$ function will returns inversely when it is applied to an negation atom. The modification of pseudo-code is shown as follows:

```

if A is unary atom then
  assuming A is in the form of  $C(?x)$ , execute query  $C(?)$ 
  if value of  $v$  is a member of query result then
    return false
  else
    return true
  end if
end if

```

4 A simple example

In this section, a simple ontology with small TBox and ABox is discussed as an example to demonstrate the reasoning process discussed in Section 3. The ontology is as follows:

TBox

AtomicClasses : *Student*; *Paper*; *Journal*

DefinedClass : *GoodStudent* \equiv *Student* \sqcap $\exists \text{write}.\exists \text{isPublishedIn}.\textit{Journal}$

ObjectProperties : *write*(*Domain* : *Student*, *Range* : *Paper*);

isPublishedIn(*Domain* : *Paper*, *Range* : *Journal*)

ABox

Student{*Mike*; *Paul*; *Jack*}; *Paper*{*P1*; *P2*; *P3*; *P4*}; *Journal*{*J1*; *J2*; *J3*; *J4*}

write{(*Mike*, *P1*); (*Mike*, *P2*); (*Paul*, *P2*); (*Paul*, *P3*); (*Jack*, *p4*)}

isPublishedIn{(*P2*, *J1*); (*P3*, *J4*)}

The reasoning task is to find out who are good students (students who wrote some papers which have been published in journal). The query for this task is written as *GoodStudent*(?). Once the task is determined, the system firstly translates the OWL-DL equivalent classes axiom which defines class *GoodStudent* to a SWRL rule:

$$\textit{Student}(?x) \wedge \textit{write}(?x, ?y) \wedge \textit{isPublishedIn}(?y, ?z) \wedge \textit{Journal}(?z) \rightarrow \textit{GoodStudent}(?x)$$

Future work will focus on evaluating the algorithm, and enriching the reasoner with more features, for example, supporting built-in functions.

References

1. Andersen, O.A., Vasilakis, G.: Building an ontology of cad model information. In: Hasle, G., Lie, K.A., Quak, E. (eds.) *Geometric Modelling, Numerical Simulation, and Optimization*, pp. 11–40. Springer (2007)
2. Baader, F., Calvanese, D., McGuinness, D.L., Nardi, D., Patel-Schneider, P.F.: *Description Logic Handbook: Theory, Implementation and Applications (2nd Edition)*. Cambridge University Press (2007)
3. Bhoopalam, K.: *Fire - A Description Logic Based Rule Engine for OWL Ontologies with SWRL-Like Rules*. Ph.D. thesis, Concordia University, Quebec, Canada (2005)
4. Donini, F.M., Lenzerini, M., Nardi, D., Schaerf, A.: Al-log: integrating datalog and description logics. *Journal of Intelligent Information Systems* 10, 227–252 (1998)
5. Hitzler, P., Krotzsch, M., Rudolph, S.: *Foundations of Semantic Web*. CRC Press (2010)
6. Horrocks, I., Patel-Schneider, P.F., Bechhofer, S., Tsarkov, D.: Owl rules - a proposal and prototype implementation. *Journal of Web Semantics: Science, Service and Agents on the World Wide Web* 3, 23–40 (2005)
7. Horrocks, I., Patel-Schneider, P.F., Boley, H., Tabet, S., Grosz, B., Dean, M.: Swrl: A semantic web rule language combining owl and ruleml. <http://www.w3.org/Submission/SWRL> (2004)
8. Horrocks, I., Patel-Schneider, P.F.: Reducing owl entailment to description logic satisfiability. In: *Journal of Web Semantics*. pp. 17–29. Springer (2003)
9. Kim, K.Y., Manley, D.G., Yang, H.: Ontology-based assembly design and information sharing for collaborative product development. *Computer-Aided Design* 38, 1233–1250 (2006)
10. Kolovski, V., Parsia, B., Sirin, E.: Extending the shoiq(d) tableaux with dl-safe rules: First results. In: Parsia, B., Sattler, U., Toman, D. (eds.) *Description Logics. CEUR Workshop Proceedings*, vol. 189. CEUR-WS.org (2006)
11. Levy, A.Y., Rousset, M.C.: Carin: A representation language combining horn rules and description logics. In: Wahlster, W. (ed.) *ECAI*. pp. 323–327. John Wiley and Sons, Chichester (1996)
12. McGuinness, D.L., Harmelen, F.V.: Owl web ontology language overview. <http://www.w3.org/TR/owl-features> (2004)
13. Mei, J., Paslaru, E.B.: Reasoning paradigms for swrl-enabled ontologies. <http://www.inf.fu-berlin.de/inst/pubs/tr-b-04> (2005)
14. Motik, B.: *Reasoning in Description Logics using Resolution and Deductive Databases*. Ph.D. thesis, University of Karlsruhe, Karlsruhe, Germany (January 2006)
15. O’Connor, M., Knublauch, H., Tu, S., Grosz, B.N., Dean, M., Grosso, W., Musen, M.: Supporting rule system interoperability on the semantic web with swrl. In: *4th International Semantic Web Conference (ISWC-2005)*. pp. 974–986 (2005)
16. Parsia, B., Sirin, E., Grau, B.C., Ruckhaus, E., Hewlett, D.: Cautiously approaching swrl. <http://www.mindswap.org/papers/CautiousSWRL.pdf> (2005)