Reasoning over OWL/SWRL Ontologies under CWA and UNA for Industrial Applications

Qingmai Wang and Xinghuo Yu

School of Electrical and Computer Engineering, RMIT University,
Melbourne, VIC 3001, Australia
qingmai.w@gmail.com, x.yu@rmit.edu.au

Abstract. As expressive schema languages, Web Ontology Language (OWL) and Semantic Web Rule Language (SWRL) have been widely introduced to many industrial applications. Most existing OWL reasoners hold Open World Assumption (OWA) and do not hold Unique Name Assumption (UNA). They lack efficiency when they are applied to industrial models which capture information under Closed World Assumption (CWA) and UNA. To overcome the problem, this paper proposes a novel backward chained ABox reasoner which efficiently reasons through OWL and SWRL under CWA and UNA.

Keywords: Ontology and rules, ABox query, Backward chaining.

1 Introduction

Ontology-based approaches have been widely used in many industrial applications. In those applications, OWL is mostly used to represent concepts and their relationships, and SWRL is usually used as an rule extension of OWL to improve the expressivity. Existing OWL reasoners, such as Pellet [11], Racer [4], and Kaon2 [7], do not make CWA and UNA. This is reasonable because OWL and SWRL originally focus on the Semantic Web (SW) in which the internet can be seen as an unlimited knowledge resource. However, when they are applied to some industrial areas where the information is usually captured under CWA and UNA, some incorrect results may be produced.

For example, OWL and SWRL have been used a lot in computer-aided Product Design and Manufacturing (PDM) to address the semantic interoperability issues [5] [12] [1] [13] [2]. In PDM, the STEP standard [9] captures information of a product under CWA and UNA. Those existing works mainly focus on how to map the STEP-based product information to the ontology while the reasoning issues are rarely mentioned. Most of them simply choose one of the general ontology reasoners for their reasoning tasks without evaluating the reasoner, whereas the reasoner may result some errors due to CWA/OWA and UNA issues, e.g., for an geometrical ontology which captures information of STEP-based product shape, existing reasoners cannot automatically find out any 4-edge face because the reasoners always assume that a face may have countless unknown edges which are not explicitly expressed in the STEP file.

This paper proposes a novel Backward Chained ABox Reasoner (BCAR) to address the above problem. The objective of BCAR is to provide efficient ABox query reasoning for ontology-based industrial models. Features of BCAR are highlighted below:

1. BCAR partly interprets OWL and SWRL under CWA and UNA (only for ABox) to improve the reasoning for closed world based industrial models.
2. Some existing OWL reasoners support SWRL by translating SWRL to some other formats (Jess, Jena or Seasame) [6] [8]. BCAR directly works on SWRL without any translation, which improves the efficiency.
3. BCAR adopts backward reasoning similar to the Prolog derivation tree. Reordering technique from Optimized Conjunctive Query (OCQ) [10] is integrated to improve the performance.

The rest of the paper is organized as follows: section 2 introduces technical details of BCAR; section 3 discusses the experiments and section 4 concludes the paper.

2 Technical Details

BCAR only focuses on ABox query reasoning for two reasons: 1)In most of the industrial applications, TBox is always decided by the valid and fixed schema of mature information models, e.g. STEP. The TBox reasoning is not necessary in this case. 2)The TBox reasoning under CWA may result inconsistence (ABox can completely determine its TBox under CWA, which may conflict with the source schema). Generally, the objective of BCAR is to help users to find out implicit instances or fillers for some defined classes or properties. For this purpose, BCAR holds following assumptions:

1. Assumption 1: The ontology contains only the explicitly expressed individuals. There is not any unknown individuals.
2. Assumption 2: Only defined classes (or properties) are allowed to have implicit instances (or fillers). Other classes (or properties) only contains instances (or fillers) that are explicitly expressed.
3. Assumption 3: If a class (or property) cannot be proved to contain an instance (or filler), then the class (or property) does not contain the instance (or filler). If a class (or property) cannot be proved to contain any instance (or filler), then the class (or property) contains nothing (negation as failure).
4. Assumption 4: Two individuals are same if and only if their names are same.

In OWL/SWRL ontologies, the most two popular methods to define classes and properties are SWRL rules and OWL Equivalent Class Axiom (ECA). BCAR only considers classes (or properties) which have corresponding ECAs or appear in heads of SWRL rules as defined classes (or properties), and the corresponding ECAs and SWRL rules are their definitions.

In the rest of this section, the concept framework of BCAR is firstly given; how to create a rule base is secondly introduced followed by details of reasoning algorithm; some special cases of reasoning are finally discussed.

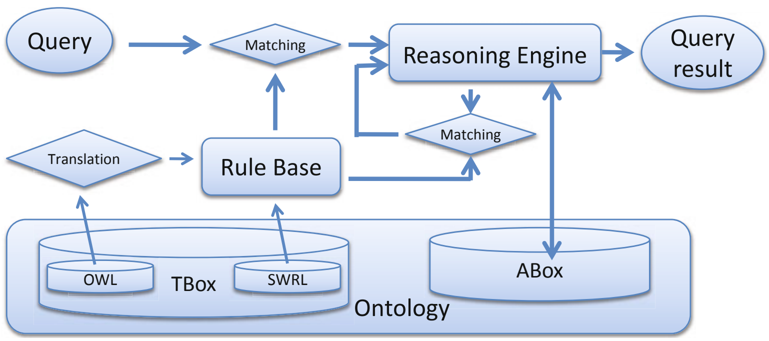


Fig. 1. Cocnept framework of BCAR

2.1 Concept Framework

Fig.1 shows the concept framework. BCAR firstly creates a unified rule base with SWRL rules and OWL ECAs. Once a query is inputted, the rule which matches the query fires. The reasoning engine is then applied to the firing rule to search for solutions for the firing rule. The engine requires information from the ABox and also may fire some other rules if required. After the searching is finished, query result is then generated based on the achieved solutions.

2.2 Building Rule Base

As mentioned above, BCAR only accepts two ways to define classes and properties: SWRL rule and ECA. Since that the backward chained reasoning engine requires a unified rule base, there is a need to translate ECAs to SWRL-like rules. The translation is generally based on the FOL semantics of the OWL constructs, and is described in Tab.1. (A, B, C are classes, P is a property, I is an individual and n is a number).

Normally, ECAs cannot be translated to rules directly since that they represent bidirectional relationships. However, BCAR consider ECAs only as class definition rules, and process ECAs equally with SWRL rules. In this case, interpret ECA as one direction logic, from definition to the class, is reasonable.

2.3 Reasoning Algorithm

In BCAR, retrieving instances (or fillers) of defined classes (or properties) fires their definition rules. The process of query is then transformed to a process of searching the ABox for solutions for the definition rule. Reasoning results are generated based on the solutions. In the following discussion, the solution is firstly defined, the searching algorithm is secondly given, how to generate results based on solutions is then described, the backward chaining in the rule base is finally discussed.

Table 1. Translating ECA to SWRL-like rules

Construct Name	ECA Syntax	Rules
owl: intersectionOf	C = A and B	A(?x) ^B(?x) -> C(?x)

owl: unionOf	C = A or B	A(?x) -> C(?x)
		B(?x) -> C(?x)

owl: complementOf	C = not A	Not (A(?x))-> C(?x)
owl: someValuesFrom	C = R some A	R(?x,?y)^A(?y) -> C(?x)
owl: allValuesFrom	C = R only A	∀ ?y(R(?x,?y)^A(?y))->C(?x)
owl: hasValue	C = R has I	R(?x, I) -> C(?x)
owl: minCardinality	C = R min *n*	(R >= *n*) (?x) ->C(?x)
owl: maxCardinality	C = R max *n*	(R <= *n*) (?x) -> C(?x)
owl: cardinality	C = R exactly *n*	(R = *n*) (?x) -> C(?x)
Complex ECA	C = A and (R some B)	R(?x,?y)^B(?y) -> H(?x) (create an intermediate class H) A(?x)^H(?x) -> C(?x)

For the purpose of simplicity, we introduce following terms to represent reasoning tasks ($\mathscr{C}$ and $\mathscr{P}$ are defined class and property respectively):

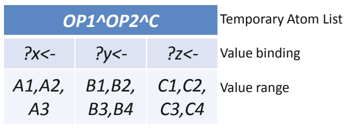
$$\mathscr{C}(?) : \text{query for all the instances of } \mathscr{C}$$
$$\mathscr{C}(? = a) : \text{check whether } a \text{ is an instance of } \mathscr{C}$$
$$\mathscr{P}(?, ?) : \text{query for all the fillers of property } \mathscr{P}$$
$$\mathscr{P}(? = a, ?) : \text{query for the instances which are the property values of } a \text{ for } \mathscr{P}$$
$$\mathscr{P}(?, ? = a) : \text{query for the instances whose property value is } a \text{ for } \mathscr{P}$$
$$\mathscr{P}(? = a, ? = b) : \text{check whether } (a, b) \text{ is a filler of property } \mathscr{P}$$

The example ontology given below is for the following algorithm discussion:

TBox
Atomic Classes:$A; B; C; D$
Defined Class:$E = OP4\ some\ C;$
Object Properties:$OP1(Domain : A, Range : B); OP2(Domain : B, Range : C, D)$
Defined Object Property:$OP4(Domain : A, Range : C)$
rule1(SWRL):$OP1(?x, ?y) \wedge OP2(?y, ?z) \wedge C(?z) \rightarrow OP4(?x, ?z)$
ABox
$A\{A1; A2; A3\}; B\{B1; B2; B3; B4\}; C\{C1; C2; C3; C4\}; D\{D1; D2\}$
$OP1\{(A1, B1); (A1, B2); (A2, B3); (A3, B3); (A3, B4)\}$
$OP2\{(B2, C2); (B2, C3); (B3, D1); (B4, C4)\}$

Solution. In a solution, all the variables in the rule are bound to a value (value can be individuals or datatype values such as integer and string), which makes all the atoms in the rule body hold true. In the example ontology, rule1 has three variables: $?x, ?y, ?z$. With the above ABox, solutions for rule1 are:

$$\{?x \leftarrow A1; ?y \leftarrow B2; ?z \leftarrow C2\}$$
$$\{?x \leftarrow A1; ?y \leftarrow B2; ?z \leftarrow C3\}$$
$$\{?x \leftarrow A3; ?y \leftarrow B4; ?z \leftarrow C4\}$$



OP1^OP2^C			Temporary Atom List
?x<-	?y<-	?z<-	Value binding
A1,A2, A3	B1,B2, B3,B4	C1,C2, C3,C4	Value range

Fig. 2. The root of searching tree

All the reasoning results of BCAR is generated based on solutions, details of which is discussed later.

Searching for Solutions. Searching for solutions is the key of BCAR. The searching process in BCAR is generally an OCQ process based on the well-known Prolog derivation tree. The idea of atoms reordering [10] is integrated to improve the performance.

Taking the example ontology, assuming the reasoning task is $OP4(?,?)$, rule1 fires. The search is then started with the following steps:

1. **Preprocessing:** Build a Temporary Atom List (TAL) from the rule body. Define a Value Range (VR) for each variable based on the Assumption 2. In the example, from rule 1 it is easy to find out that $?x$ belongs to A, $?y$ belongs to B, and $?z$ belongs to C. Based on Assumption 2, $?x$, $?y$ and $?z$ can only be the explicitly asserted members of A, B and C respectively. Fig.2 shows the TAL and VRs of the example. Other cases are listed below:
   (a) The VR of $?x$ (or $?y$) contains only $A1$ (or $B1$) if the reasoning task is $OP4(? = A1, ?)$ (or $OP4(?, ? = B1)$);
   (b) For constant, BCAR create new variables whose VR only contain the constant's value;
   (c) For variable belonging to a defined class, the VR $= \top$ (Assumption3);
   (d) For datatype variable, the VR $= \infty$
2. **Variable choosing and branching:** The performance of CQ heavily relies on the query ordering. In BCAR, the reasoning always start from the variable which has minimal size of VR and have not been bound to a value, so that the number of branches of the searching tree is minimized. The selected variable is called SV. In the example, $?x$ is SV since it has minimal size of VR among all the variables. BCAR then generate branches for each value in VR of $?x$.
3. **Binding and intersecting:** In each branch, the SV is bound to a value. BCAR then processes the atoms related to the SV in TAL, which may reduce the VR of SV or other variables which are related to SV by an intersecting process. After that, the processed atoms are removed from TAL. In the example, $?x$ is bound to $A1$ in the first branch. BCAR then processes $OP1(?x(= A1), ?y)$ which is the only atom related to $?x$ in TAL. Based on the ABox and Assumption 3, $?y$ can only be either $B1$ or $B2$. BCAR then intersects $\{B1, B2\}$ with $?y$'s original VR $\{B1, B2, B3, B4\}$ to be the new VR of $?y$, as Fig.3 shows:

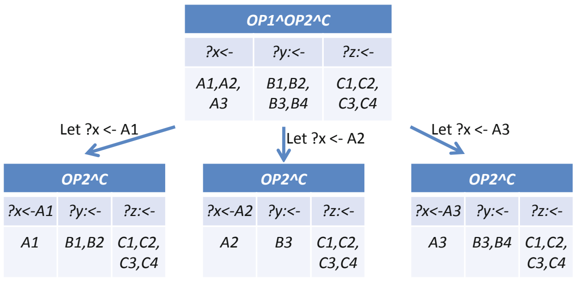


Fig. 3. Generating branches

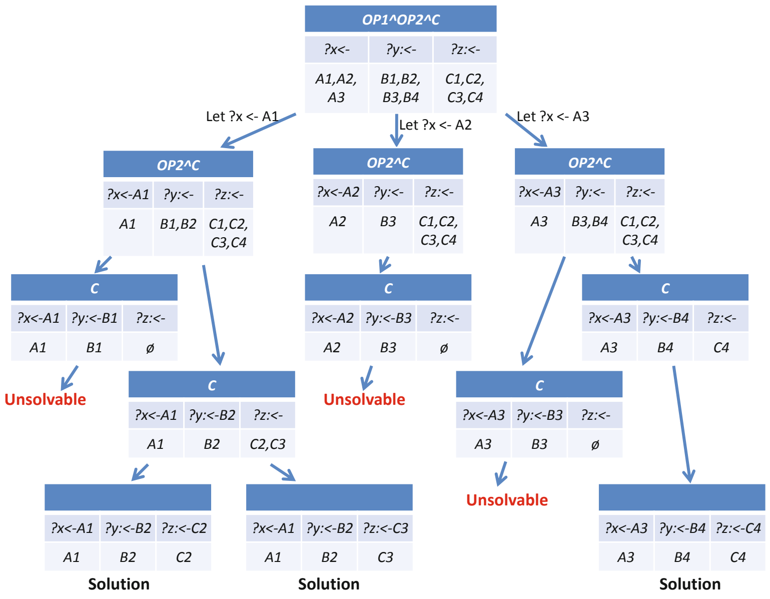


Fig. 4. Searching tree

4. **Termination:** BCAR repeats the above step 2 and step 3 until: 1)VR of any variable turns out to be empty (based on Assumption 1&2, it means this variable is unsolvable in this branch); 2)All the variables have been bound to a value (a solution has been found in this branch); Fig4. shows how the search tree find solutions and how it is terminated.

Generating Reasoning Results. With solutions, BCAR generate results for every reasoning tasks as follows (assuming the head of the definition rule of $\mathscr{C}$ and $\mathscr{P}$ are $\mathscr{C}(?x)$ and $\mathscr{P}(?x, ?y)$ respectively):

1. $\mathscr{C}(?)$: return all the values of $?x$ from all the solutions;
2. $\mathscr{C}(? = a)$: set the initial VR of $?x$ to be $\{a\}$, check wether a solution can be found;

3. $\mathscr{P}(?,?)$: pick up all the values of $?x$ and $?y$ from all the solutions;
4. $\mathscr{P}(?=a,?)(or\mathscr{P}(?,a))$: set the initial VR of $?x$ (or $?y$) to be $\{a\}$, return all the values of $?y$ (or $?x$) from all the solutions;
5. $\mathscr{P}(?=a,?=b)$: set the initial VR of $?x$ and $?y$ to be $\{a\}$ and $\{b\}$ respectively, check wether a solution can be found;

Backward Chaining. In the "Binding and intersecting" step of the searching process, if BCAR need to process an atom corresponding to a defined class or property, another rule fires. For example, considering defined class $E = OP4\ some\ A$, from Tab.1 the translation rule is $OP4(?x,?y) \wedge D(?y) \rightarrow E(?x)$. Assuming the reasoning task (goal) is $E(?=A1)$, when searching for the solutions for the translation rule, the "Binding and intersecting" step will add another reasoning task (new goal) $OP4(?=A1,?)$ and rule 1 consequently fires. This is so-called backward chained reasoning.

2.4 Handling Special Rules and Atoms

In Tab.1, some of the translation rules can be solved normally using the above algorithm (e.g. owl:intersectionOf; owl:unionOf; owl:someValuesFrom), while the others can not. This section generally discusses how BCAR handles some special rules and atoms.

owl:complementOf. $C = notA$ is translated to $Not(A(?x)) \rightarrow C(?x)$. Based on Assumption 3 (negation as failure), BCAR handles the translation rule as follows:

1. $C(?)$: return the individuals which cannot be proved to be instances of A;
2. $C(?=a)$: return false if a is proved to be an instance of A, otherwise return true;

owl:allValuesFrom. $C = R\ only\ A$ is translated to $\forall ?y(R(?x,?y) \wedge A(?y)) \rightarrow C(?x)$. Based on Assumption 2, BCAR handles the translation rule as follows:

1. $C(?)$: return the individuals which are proved to have some property values for R and all these values are proved to be instances to A;
2. $C(?=a)$: return true if a is proved to have some property values for R and all these values are proved to be instances of A, otherwise return false;

owl:minCardinality/maxCardinality/cardinality. $C = min$ or max or $exactly\ a$ is translated to $(R \geq$ or $\leq$ or $= a)(?x) \rightarrow c(?x)$. Based on Assumption 2, BCAR handles the translation rule as follows:

1. $C(?)$: return the individuals which are proved to have more than or less than or exactly a different property values for R;
2. $C(?=a)$: return true if a is proved to have more than or less than or exactly a property values for R, otherwise return false;

Complex ECA. A complex ECA is a combination of basic OWL constructs. As Tab.1 shows, BCAR transform complex ECA to multiple basic rules with self-created intermediate classes between them.

swrl:differentFrom/sameAs. An SWRL rule may contains atoms such as "differentFrom(?x,?y)" and "sameAs(?x,?y)". In this case, BCAR firstly remove these atoms from TAL before searching for solutions. After solutions are found, these comparison atoms are used to validate each solution based on Assumption 4. Only validated solutions are outputted in the end.

swrl:built-in. BCAR only supports built-Ins for comparison in current stage. BCAR process built-In atom similar to the "differentFrom/sameAs" atoms. The only difference is "differentFrom/sameAs" atoms focus on comparing individuals while built-In atoms focus on comparing datatype values.

3 Experiments

In this section, two experiments has been done to test the performance of searching and the effectiveness of BCAR in industrial applications.

The first experiment tests BCAR with LUBM[1,0] [3] which contains 103074 triples, and compares the performance with popular reasoner Pellet and RacerPro. The 14 standard quires are transferred to rules for BCAR. An example of transfer is shown below:

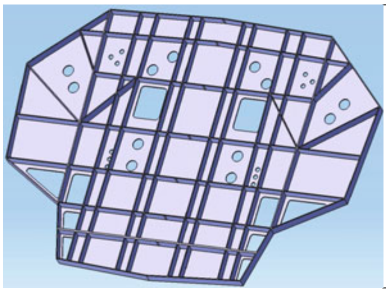
$$ub : GraduateStudent(?x) \wedge ub : takesCourse(?x, http : \\ //www.Department0.University0.edu/GraduateCourse0) \rightarrow Query01(?x)$$

As mentioned before, in BCAR, generating results for $Query01(?)$ is the process of searching for solutions for rule body. And searching for solutions is essentially a conjunctive query of body atoms. In this way, the 14 standard quires of LUBM are used to test the searching performance of BCAR.

Table 2. Experimental result (Unit: ms)

	Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8	Q9	Q10	Q11	Q12	Q13	Q14
BCAR	<1	340	4	<1	112	670	12	966	380	1	N/A	6	7	660
Pellet	78	536	2	8	34	97	11	1102	867	3	35	38	2	88
RacerPro	308	896	24	201	66	899	1033	1297	1156	92	43	881	306	630

The experimental result is shown in Tab.2. Currently BCAR doesn't support transitive roles, so no result can be found for Query 11. Generally BCAR shows a decent performance comparing with other two reasoners. Especially for quires which require only small part of ABox information, BCAR performs very



Rule1: face(? x) ∧ face_has_face_bounds(? x, ? y)
∧ face_edge(? y)
∧ face_bound_has_bound(? y, ? z)
∧ edge_loop_has_edges(? z, ? m)
→ face_has_edges(? x, ? m)
Rule2: face(? x) ∧ face(? y) ∧ face_has_edges(? x, ? z)
∧ face_has_edges(? y, ? z)
∧ differentFrom(? x, ? y)
→ face_has_neighbor_faces(? x, ? y)
Rule3: edge(? x) ∧ edge_has_angle(? x, ? y)
∧ smaller(? y, 180)
→ concave_angular_edge(? x)
ECA1: face ⊓ ∀face_has_edges. concave_angular_edge
≡ pocket_bottom_face
ECA2: face ⊓ ∃face_has_heighbor_face. pocket_bottom_face
≡ pocket_wall_face

Fig. 5. Input model and rules

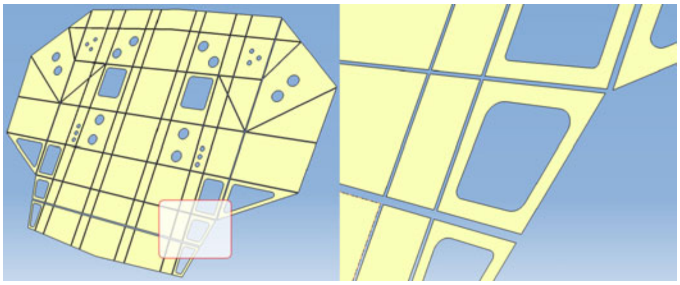


Fig. 6. Visualization of query result

good since that backward reasoning guarantees only necessary information is processed.

Another experiment tests the effectiveness of applying BCAR to an industrial model as shown in the left of Fig.5. The model comes from a STEP file which captures the geometric information of a product. After transferring from STEP to OWL, the ontology contains classes such as faces, edges and points. Instances of the STEP file are mapped to OWL individuals. Several SWRL rules and OWL ECAs (shown in right of Fig.5) are added to the ontology to define a class *pocket_bottom_face*, and BCAR is applied to find out all the members of *pocket_bottom_face*. The visualization of query result is shown in Fig.6.

Other reasoners are not applicable in this case for the UNA and OWA/CWA issues. For example, without making UNA the atom $differentFrom(?x, ?y)$ in Rule2 always returns false because the STEP file doesn't explicitly mention that an instance is different from another instance. Similarly, the universal quantification in ECA1 always returns false, because under OWA the reasoners always believe there must exist some unknown edges.

4 Conclusion

This paper has proposed a novel Backward Chained ABox reasoner which provides efficient ABox query reasoning under CWA and UNA for industrial applications. The reasoner firstly translated OWL ECAs to SWRL-like rules, and then backward reason through the unified rule base of the ontology to retrieval instances for query tasks.

Experiments shows that the BCAR can effectively execute queries for industrial models under CWA and UNA, and also demonstrates that it has a decent performance. Future works will focus on improving the reasoner to support more OWL constructs and evaluating the reasoner using more practical industrial cases.

References

1. Abdul-Ghafour, S., Ghodous, P., Shariat, B., Perna, E.: Towards an intelligent cad models sharing based on semantic web technologies. In: Curran, R., Chou, S.-Y., Trappey, A. (eds.) Collaborative Product and Service Life Cycle Management for a Sustainable World. Advanced Concurrent Engineering, pp. 195–203. Springer, Heidelberg (2008)
2. 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, Heidelberg (2007)
3. Guo, Y., Pan, Z., Heflin, J.: Lubm: A benchmark for owl knolwedge base systems. Journal of Web Semantics 3, 158–182 (2005)
4. Haarslev, V., Moller, R.: Racer: An owl reasoning agent for the semantic web. In: Proceedings of the International Workshop on Applications, Products and Services of Web-based Support Systems, pp. 91–95 (2003)
5. 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)
6. Mei, J., Paslaru, E.B.: Reasoning paradigms for swrl-enabled ontologies (2005), `http://www.inf.fu-berlin.de/inst/pubs/tr-b-04`
7. Motik, B.: Reasoning in Description Logics using Resolution and Deductive Databases. Ph.D. thesis, University of KarLsruhe, Karlsruhe, Germany (January 2006)
8. O'Connor, M.F., Knublauch, H., Tu, S., Grosof, B.N., Dean, M., Grosso, W., Musen, M.A.: Supporting Rule System Interoperability on the Semantic Web with SWRL. In: Gil, Y., Motta, E., Benjamins, V.R., Musen, M.A. (eds.) ISWC 2005. LNCS, vol. 3729, pp. 974–986. Springer, Heidelberg (2005)
9. Owen, J.: STEP: An Introduction. Information Geometers (1997)
10. Sirin, E., Parsia, B.: Optimizations for answering conjunctive abox queries: First results. In: Proc. of the Int. Description Logics Workshop, DL (2006)
11. Sirin, E., Parsia, B., Grau, B.C., Kalyanpur, A., Katz, Y.: Pellet: A practical owl-dl reasoner. Web Semantics: Science, Services and Agents on the World Wide Web 55, 51–53 (2007)
12. Yang, D., Dong, M., Miao, R.: Development of a product configuration system with an ontology-based approach. Computer-Aided Design 40, 863–878 (2008)
13. Zhao, W., Liu, J.K.: Owl/swrl representation methodology for express-driven product information model: Part 1. implementation methodology. Computers in Industry 59, 580–589 (2008)