IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received January 16, 2021, accepted January 25, 2021, date of publication February 2, 2021, date of current version February 26, 2021.

Digital Object Identifier 10.1109/ACCESS.2021.3056556

Extracting Satisfiability-Preserving Modules From the OWL RL Ontology for Efficient Reasoning

XIAOFEI ZHAO[1,2], FANZHANG LI[2], AND HONGJI YANG[3], (Member, IEEE)

[1]School of Computer Science and Technology, Tiangong University, Tianjin 300387, China
[2]Provincial Key Laboratory for Computer Information Processing Technology, Soochow University, Suzhou 215006, China
[3]School of Informatics, University of Leicester, Leicester LE1 7RH, U.K.

Corresponding author: Xiaofei Zhao (zhaoxiaofei1978@hotmail.com)

This work was supported in part by the National Natural Science Foundation of China (NSFC) under Grant 61972456, and in part by the Open Foundation of Jiangsu Provincial Key Laboratory for Computer Information Processing Technology under Grant KJS1737.

ABSTRACT Reasoning on large ontologies has been identified as an important challenge. Concept satisfiability is one of the core reasoning tasks in ontology engineering. To improve the efficiency of concept satisfiability checking for OWL RL ontologies, we propose an approach for extracting modules from the OWL RL ontology. Our approach is based on the idea that for a given concept being checked, not all of the ontology elements are related to the checking process. By transforming the OWL RL axioms into first-order constraints and analyzing the dependencies among the constraints, we give a strategy for deleting the irrelevant elements; thus, a simplified ontology that preserves the original checking result can be obtained. The experimental results obtained for a set of real-world ontologies show that our method can effectively reduce the reasoning time, and the efficiency is increased by 1.15 times to 9.08 times.

INDEX TERMS OWL RL, ontology reasoning, module extraction, first-order constraint.

I. INTRODUCTION

As the Web Ontology Language (OWL) recommended by the Worldwide Web Consortium (W3C) for the unified description of information resources in a Web environment, OWL 2[1] enables the description of hierarchies among concepts and the semantics of concepts and attributes through a series of vocabularies with explicit semantics, thus becoming the foundation for constructing semantic Web ontologies. Among various other improvements, OWL 2 is the first that adequately addresses the trade-off between logical expressivity and scalability that is inherent to formal knowledge representation by specifying additional language profiles. These profiles are OWL EL based on EL++ [2], OWL QL based on DL-Lite [3], [4] and OWL RL [5], [6], which was designed with rule-based implementations in mind.

The OWL RL profile imposes restrictions on the use of OWL 2 constructs so that reasoning tasks can be implemented as a set of rules in a forward-chaining reasoning engine. It is realized as a partial axiomatization of the OWL 2 semantics in the form of first-order implications inspired by description logic programs (DLPs) and pD∗, which gives it desirable computational properties for implementation using a standard rule language.

Standard reasoning tasks involved in the OWL 2 ontology include concept satisfiability, classification, and consistency. Since the last two tasks can be reduced to the first one, concept satisfiability checking has been identified as a core reasoning task. A class C_1 is satisfiable if there is an OWL objectification that satisfies all axioms and contains at least one object of C_1. Although standard reasoning tasks are supported by state-of-the-art OWL 2 reasoners, such as HermiT [7], Fact++ [8] and Pellet, reasoning on large OWL RL ontologies is an important challenge, especially in the case of large rule sets. Improving the efficiency of OWL RL ontology reasoning is therefore an urgent task.

To perform reasoning on a subontology with a smaller scale, the notion of a module, i.e., a property-preserving subontology, has been proposed. Module extraction has received much attention in recent years [9]–[11], [25]–[27]. Originally motivated by ontology reuse [28], [29], ontology modularity has been widely used in different areas, such as ontology matching [30] and debugging [31], forgetting [32], [33], or to improve reasoning [34]. In this article, we focus on applications in ontology reasoning. The increase in expressiveness can make the problem of optimal module extraction hard. As a consequence, most existing extraction algorithms

The associate editor coordinating the review of this manuscript and approving it for publication was Shuihua Wang.

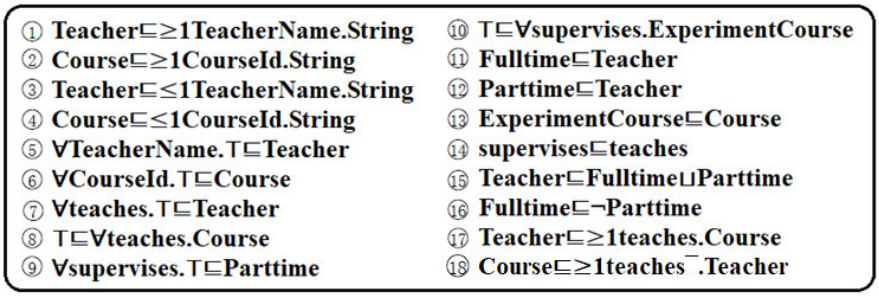


FIGURE 1. OWL RL ontology for the relations between teachers and courses.

can only deal with light-weight ontology languages. Various methods have been proposed for ontology languages $\mathcal{ELI}$ [12], $\mathcal{ALCQI}$ [13], $\mathcal{ELH}$ [14] and RDFS [15], [16]. To the best of our knowledge, extracting modules from the OWL RL ontology has not been thoroughly studied thus far.

The OWL RL ontology is shown in Fig. 1, which describes the relationship between a *Teacher* and *Course* in a school. To concisely express the semantics of OWL RL axioms, we use description logic syntax throughout this article. Axiom 1 and axiom 3 are the restrictions DataMinCardinality and DataMaxCardinality for the data property *TeacherName*, and similar restrictions for the data property *CourseId* are specified by axiom 2 and axiom 4. DataPropertyDomain for *TeacherName* and *CourseId* are specified by axioms 5∼6. ObjectPropertyDomain and ObjectPropertyRange for object properties *teaches* and *supervises* are specified by axioms 7∼10. Axioms 11∼13 state that *Teacher* has subclasses *Fulltime* and *Parttime*, and *ExperimentCourse* is a subclass of *Course*. Axiom 14 states that *supervises* is a subproperty of *teaches*. The last four axioms ensure that a teacher belongs to the set of either full-time teachers or part-time teachers (axiom 15), the set of full-time teachers and the set of part-time teachers are disjoint (axiom 16), a teacher must teach at least one course (axiom 17), and a course must be taught by at least one teacher (axiom 18).

To improve the efficiency of OWL RL ontology reasoning, in this article, we propose a method for extracting a satisfiability-preserving module from the OWL RL ontology. The main idea of our method is as follows: for a given concept $\mathcal{C}_{\mathcal{M}}$ to be checked, not all of the ontology elements are related to the checking process. In fact, only the axioms that may conflict with $\mathcal{C}_{\mathcal{M}}$ and the classes, data properties, and object properties involved in these axioms are related to the checking process; thus, other classes, data properties, object properties and axioms can be deleted from the ontology while maintaining the satisfiability checking result of $\mathcal{C}_{\mathcal{M}}$. It is obvious that the reasoning efficiency will be improved because the checking is carried out on a simplified ontology that contains only the related elements. Our method is based on a strict formalization mechanism. The precise proofs for every reduction strategy are given on the OWL RL ontology.

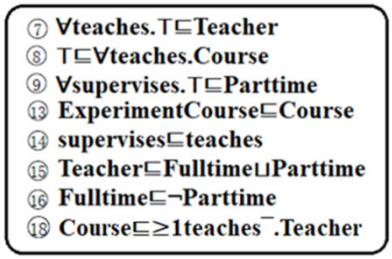


FIGURE 2. Module for checking the satisfiability of *Teacher* ⊓ (∃*supervises*.*ExperimentCourse*).

Take Fig. 1 as an example; if the goal is to check the satisfiability of the concept *Teacher* ⊓ (∃ *supervises*. *Experiment Course*), 10 of the 18 axioms in the original ontology will be deleted during the extraction process, and we will obtain the module that is shown in Fig. 2. This module is significantly smaller than the original ontology. To further verify the effectiveness of our method, we conducted many tests. The experimental results show that the checking efficiency can be significantly improved through the extraction, and the efficiency is up to 9.08 times greater in the best case.

II. THE LOGICAL FOUNDATION OF CONSTRAINT DEPENDENCY ANALYSIS AND MAIN IDEA OF OUR APPROACH

A. THE LOGICAL FOUNDATION OF CONSTRAINT DEPENDENCY ANALYSIS

In what follows, we give a brief introduction to a logical mechanism that is closely related to this article.

Both variables and constants are called terms, denoted as t. The atom $p(t_1, \ldots, t_n)$ is a predicate that acts on n terms, $t_1, \ldots, t_n$, denoted as $p(\bar{t})$. If all the terms in an atom are constants, the atom is called a ground atom. Both an atom $p(\bar{t})$ and a negative atom $\neg p(\bar{t})$ are called literals, denoted as ℓ. A ground atom of predicate p is called an object of p. The finite set of objects of one or more predicates is called an objectification. The substitution π is a set of the form $\{x_1/t_1, \ldots, x_n/t_n\}$ in which each variable x_i is different. The set of all x_i is called the domain of the substitution π, denoted as $domain(\pi)$. If all t_i are constants, the substitution π is called a ground substitution. $\ell\pi$ denotes the new literal

that results from synchronously replacing every x_i in the literal ℓ with the corresponding t_i through π. Similarly, $\varphi\pi$ denotes the new conjunction that results from synchronously applying π to every literal in the conjunction φ. Given the conjunction φ of the literals, the predicate p and the variable x, $assignmentupper(\varphi, p, x)$ and $assignmentlower(\varphi, p, x)$ denote the maximal and minimal number of ground atoms of p, respectively, that need to be assigned to ensure that every literal in p containing x is true. For example, when $\varphi = p(x,u) \wedge p(x,v) \wedge p(x,w) \wedge u \neq v$, we obtain that $assignmentupper(\varphi, p, x) = 3$ because $u \neq v, v \neq w$ and $u \neq w$, while $assignmentlower(\varphi, p, x) = 2$ because $w = u$ or $w = v$.

The *denial constraint* is a kind of rule of the form $\forall \bar{t}.\varphi(\bar{t}) \rightarrow \perp$, where φ is the conjunction of the literals and $\perp$ is an atom that is false. Roughly, the left side of a denial constraint denotes the conditions that will never be satisfied by any objectification. An *embedded dependency* (*ED*) is a kind of rule of the form $\forall \bar{t}.\varphi(\bar{t}_\varphi) \rightarrow \bigvee_{i=1\ldots n} \exists \bar{y}_i.\phi_i(\bar{t}_i, \bar{y}_i)$, where n is a nonnegative integer. When $n = 0$, the right side of the *ED* is empty. Therefore, the denial constraint is essentially a special form of *ED*.

Given an objectification O, if O satisfies the left side of an *ED* but does not satisfy its right side, we say that O violates the *ED*; i.e., there is some ground substitution π such that $O \models \varphi(\bar{t}_\phi)\pi$, but there is not any ground substitution σ such that $O \models \phi_i(\bar{t}_i, \bar{y}_i)\pi\sigma$ for some ϕ_i. If objectification O violates an *ED*, repairing this *ED* means constructing a superset O' of O. O' contains the necessary objects of the predicate on the right side of this *ED,* and for any ground substitution π, if $O' \models \varphi(\bar{t}_\phi)\pi$, then there must be some ground substitution σ such that for some ϕ_i, we have $O \models \phi_i(\bar{t}_i, \bar{y}_i)\pi\sigma$.

Given the set $\mathcal{D}$ of *EDs* and an objectification O, we say that O is consistent with regard to $\mathcal{D}$ (denoted as $O \models \mathcal{D}$) if O does not violate any *ED* in $\mathcal{D}$. If there is at least one consistent objectification O with regard to $\mathcal{D}$, $\mathcal{D}$ is said to be satisfiable.

B. THE MAIN IDEA OF OUR APPROACH

Given an OWL RL ontology and a concept $\mathcal{C}_\mathcal{M}$ to be checked, extracting a satisfiability-preserving module for $\mathcal{C}_\mathcal{M}$ means deleting axioms from the original ontology while maintaining the satisfiability result of $\mathcal{C}_\mathcal{M}$. To achieve this goal, we formally describe the OWL RL ontology and $\mathcal{C}_\mathcal{M}$ by means of first-order logic. The OWL RL axioms and $\mathcal{C}_\mathcal{M}$ are translated into a set of first-order constraints, while the task of checking $\mathcal{C}_\mathcal{M}$ satisfiability is transformed into a satisfiability check of a first-order formula. Specifically, the classes, the data properties and the object properties are formalized as first-order predicates, while the axioms and $\mathcal{C}_\mathcal{M}$ are formalized as first-order formulas. For example, the task of checking the satisfiability of *Course* is transformed into a satisfiability check of $Course(c)$, while the task of checking whether there is a *Course* with a teacher who is not a full-time teacher is a satisfiability check of $Course(c) \wedge teaches(t, c) \wedge \neg Fulltime(t)$.

Because the process of satisfiability checking for the first-order formula corresponding to $\mathcal{C}_\mathcal{M}$ is the process of searching for objectifications that satisfy all the constraints as well as $\mathcal{C}_\mathcal{M}$, the checking results will not be changed if we can identify the constraints that may make $\mathcal{C}_\mathcal{M}$ unsatisfiable and then delete the other constraints from the ontology. Based on this intuition, our goal becomes distinguishing which constraints may make $\mathcal{C}_\mathcal{M}$ unsatisfiable.

Next, we will analyze the cases in which certain constraints may make $\mathcal{C}_\mathcal{M}$ unsatisfiable in order to provide the basis for deleting these constraints. As mentioned earlier, the process of satisfiability checking for the first-order formula corresponding to $\mathcal{C}_\mathcal{M}$ is the process of searching the objectifications that satisfy all the constraints as well as $\mathcal{C}_\mathcal{M}$; in other words, to decide whether $\mathcal{C}_\mathcal{M}$ is satisfiable is to decide whether we could repair an objectification that satisfies $\mathcal{C}_\mathcal{M}$. Suppose that there is an objectification O_1 satisfying $\mathcal{C}_\mathcal{M}$; we cannot decide the satisfiability of $\mathcal{C}_\mathcal{M}$ by only using O_1 if O_1 violates some constraints. However, we can repair the constraints that are violated by inserting the necessary new objects into O_1 with the prerequisite of satisfying $\mathcal{C}_\mathcal{M}$. This process is iterative because the insertion of the new objects may cause other constraints to be violated. We iteratively insert the new objects to repair the constraints that are violated until either we obtain an objectification consistent with all the constraints or there are inconsistent constraints remaining and we cannot continue to repair. The former implies that $\mathcal{C}_\mathcal{M}$ is satisfiable, and the latter implies that it is unsatisfiable.

From the above analysis, we can see that the constraints that may make $\mathcal{C}_\mathcal{M}$ unsatisfiable fall into the following two categories: (1) the constraints that will be violated when we try to satisfy $\mathcal{C}_\mathcal{M}$(or to repair the violated constraints that follow) and (2) the constraints that will violate other constraints when they are repaired. Taking Fig. 1 as an example, suppose that we want to decide the satisfiability of *Course*. An objectification that contains only one object of *Course* satisfies *Course* but violates the constraint that a *Course* should be taught by at least one *Teacher*. To repair this constraint, we need to insert an object of *Teacher* related to this *Course*. However, this insertion violates the constraint that a teacher is either full-time or part-time. Therefore, the first constraint may make *Course* unsatisfiable, as it will be violated when we try to satisfy $\mathcal{C}_\mathcal{M}$, and its repair will violate other constraints. The second constraint will never make *Course* unsatisfiable, as whenever it is violated, we can repair it using the object of *Fulltime* to replace that of *Teacher*, and such a repair will not violate any other constraints. We establish a constraint dependency graph to record such relationships between the constraints. It should be emphasized that the purpose of this article is not to infer the satisfiability of $\mathcal{C}_\mathcal{M}$ but to extract a satisfiability-preserving subontology for $\mathcal{C}_\mathcal{M}$. The analysis of the above iterative process is performed only to clarify the cases to which the constraints that may make $\mathcal{C}_\mathcal{M}$ unsatisfiable

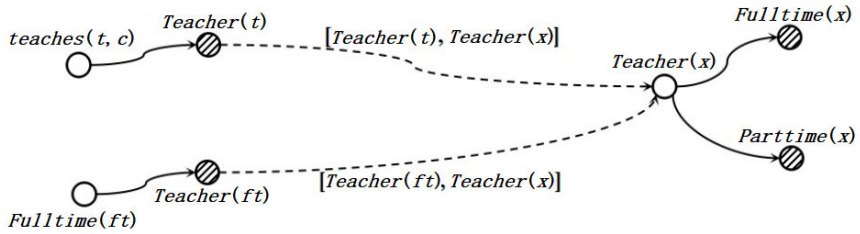


FIGURE 3. An example of repair/violation interactions between *EDs*.

belong. In Section 5, we will give a reduction strategy for each case.

Our approach is divided into three steps. The first step is to transform the OWL RL axioms into a set of first-order constraints expressed as *EDs* (Section 3). The second step is to establish a constraint dependency graph based on these *EDs* (Section 4). The third step is to put $\mathcal{C}_{\mathcal{M}}$ into the constraint dependency graph and execute the reduction strategy and then to construct the module for $\mathcal{C}_{\mathcal{M}}$ (Section 5).

III. TRANSFORMING OWL RL AXIOMS INTO FIRST-ORDER CONSTRAINTS

To accurately reflect the interactions among the OWL RL axioms and distinguish the constraints to be deleted, we need a well-defined formalization mechanism. We conduct the formalization by transforming each OWL RL axiom into an *ED*.

First, a single first-order predicate is used to describe every class, data property and object property. Taking Fig. 1 as an example, we obtain the following predicates: *Teacher*(t), *Course*(c), *Fulltime(ft)*, *Parttime(pt)*, *ExperimentCourse(ec)*, *TeacherName*(t, n), *CourseId*(c, i), *teaches*(t, c), *supervises(pt, ec)*.

Then, based on the semantics of the OWL 2 RL/RDF rules described in [17], each axiom is transformed into one denial constraint (two in the case of EquivalentClasses and EquivalentObjectProperties). For example, DataMaxCardinality for the data property *CourseId* in Fig. 1 (axiom 4), i.e., "the upper bound cardinality of *CourseId* is 1," is transformed into $Course(c) \wedge CourseId(c, i_1) \wedge CourseId(c, i_2) \wedge i_1 <> i_2 \rightarrow \bot$. This formula states that if *Course* c has different *id*s i_1 and i_2 simultaneously, then the constraint is violated. As further examples, axiom 15 is transformed into $Teacher(t) \wedge \neg(Fulltime(t) \vee Parttime(t)) \rightarrow \bot$ and axiom 16 into $Fulltime(t) \wedge Parttime(t) \rightarrow \bot$. Through the above formalization, all the axioms can be accurately described as denial constraints.

Next, we transform each denial constraint into an equivalent *ED* by moving all the negative literals from the left side of each constraint to the right side and deleting the negative symbols. For example, $Teacher(t) \wedge \neg(Fulltime(t) \vee Parttime(t)) \rightarrow \bot$ is transformed equivalently into $Teacher(t) \rightarrow Fulltime(t) \vee Parttime(t)$. Now, we change every variable of the atoms in every *ED* to a different name so that substitutions can be applied when we match the different *EDs* while establishing the constraint dependency graph.

The correspondence between the OWL RL axioms and *EDs* is given in Tab. 1.

In accordance with Tab. 1, the *ED* corresponding to each axiom in Fig. 1 is listed in Tab. 2.

IV. ESTABLISHING A CONSTRAINT DEPENDENCY GRAPH BASED ON EDS

In what follows, we will establish the interaction between a repair applied to satisfy a constraint and the violation of other constraints resulting from applying such a repair. As mentioned in Section 2.1, an *ED* is violated by an objectification if the objectification satisfies its left side but does not satisfy its right side; so, the left side of each *ED* is the prerequisite for violating the *ED* (here, we use the word "prerequisite" to show that the *ED* is not necessarily violated), while every conjunction on the right side of the *ED* is a specific method to repair it (since the insertion of a new object of every conjunction into the objectification can repair a violation when it occurs). Furthermore, if the right side of an *ED* and the left side of another *ED* contain the same atom, repairing the former *ED* by inserting a new object of this atom may cause the latter to be violated (since the left side of the latter *ED* may be satisfied after the insertion); hence, there is a repair/violation interaction between two such *EDs*. We record such interactions using a constraint dependency graph.

The nodes in a constraint dependency graph are divided into two categories: constraint nodes and repair nodes. The left side of each *ED* corresponds to a constraint node (represented by a hollow circle). Each conjunction on the right side of each *ED* corresponds to a repair node (represented by a solid circle). Repair/violation interactions between *EDs* are represented by directed dotted edges. For example, the interactions among *ED* 7, *ED* 11 and *ED* 15 in Tab. 2 are shown in Fig. 3. Since the insertion of an object of *Teacher* to repair *ED* 7 may violate *ED* 15, we draw a directed dotted edge from the repair node *Teacher*(t) of *ED* 7 to the constraint node *Teacher*(x) of *ED* 15 and mark the edge as [*Teacher*(t), *Teacher*(x)] to record the shared atom of the right side of *ED* 7 and the left side of *ED* 15. Similarly, we draw a directed dotted edge from the repair node of *ED* 11 to the constraint node of *ED* 15.

Now, we give the formal definition of a constraint dependency graph.

TABLE 1. Correspondence between OWL RL axioms and *EDs*.

OWL RL axiom	*ED*
$A \sqsubseteq B$	$A(x) \rightarrow B(x)$
$R \sqsubseteq S$	$R(x,y) \rightarrow S(x,y)$
$A \equiv B$	$A(x) \rightarrow B(x)$, $B(x) \rightarrow A(x)$
$R \equiv S^{-}$	$R(x,y) \rightarrow S(y,x)$, $S(x,y) \rightarrow R(y,x)$
$A \sqsubseteq \neg B$	$A(x) \wedge B(x) \rightarrow \perp$
$A \sqsubseteq B_1 \sqcap B_2$	$A(x) \rightarrow B_1(x) \wedge B_2(x)$
$A \sqsubseteq B_1 \sqcup B_2$	$A(x) \rightarrow B_1(x) \vee B_2(x)$
$A \sqsubseteq \exists R.B$	$A(x) \rightarrow R(x,y) \wedge B(y)$
$A \sqsubseteq \exists R.DateType$	$A(x) \rightarrow R(x,y)$
$A \sqsubseteq \exists R.\{a\}$	$A(x) \rightarrow R(x,a)$
$\forall R.\top \sqsubseteq A$	$R(x,y) \rightarrow A(x)$
$\top \sqsubseteq \forall R.A$	$R(x,y) \rightarrow A(y)$
$A \sqsubseteq \forall R.B$	$A(x) \wedge R(x,y) \rightarrow B(y)$
$A \sqsubseteq \forall R.\{a\}$	$A(x) \wedge R(x,y) \wedge y <> a \rightarrow \perp$
$A \sqsubseteq \leq nR.B$	$A(x) \wedge (\wedge_{i=1}^{n+1} B(y_i)) \wedge (\wedge_{i=1}^{n+1} R(x, y_i)) \wedge y_1 <> y_2 \wedge \ldots \wedge y_1 <> y_{n+1} \wedge \ldots \wedge y_n <> y_{n+1} \rightarrow \perp$
$A \sqsubseteq \leq nR.DataType$	$A(x) \wedge (\wedge_{i=1}^{n+1} R(x, y_i)) \wedge y_1 <> y_2 \wedge \ldots \wedge y_1 <> y_{n+1} \wedge \ldots \wedge y_n <> y_{n+1} \rightarrow \perp$
$A \sqsubseteq \geq nR.B$	$A(x) \rightarrow (\wedge_{i=1}^{n} B(y_i)) \wedge (\wedge_{i=1}^{n} R(x, y_i)) \wedge y_1 <> y_2 \wedge \ldots \wedge y_1 <> y_n \wedge \ldots \wedge y_{n-1} <> y_n$
$A \sqsubseteq \geq nR.DataType$	$A(x) \rightarrow (\wedge_{i=1}^{n} R(x, y_i)) \wedge y_1 <> y_2 \wedge \ldots \wedge y_1 <> y_n \wedge \ldots \wedge y_{n-1} <> y_n$
$R_1 \circ \ldots \circ R_m \sqsubseteq R$	$\wedge_{i=1}^{m} R_i(x_i, x_{i+1}) \rightarrow R(x_1, x_{m+1})$
Functional(R)	$R(x,y_1) \wedge R(x,y_2) \wedge y_1 <> y_2 \rightarrow \perp$
Irreflexive(R)	$R(x,x) \rightarrow \perp$
Asym(R)	$R(x,y) \wedge R(y,x) \rightarrow \perp$
Disjoint(R_1,R_2)	$R_1(x,y) \wedge R_2(x,y) \rightarrow \perp$

Definition 1 (Constraint Dependency Graph). *A constraint dependency graph G is a directed bipartite graph denoted as a 6-tuple <CN, RN, E_S, E_P, h, g> that satisfies the following:*

(1) CN is the set of constraint nodes.

(2) RN is the set of repair nodes.

(3) E_S is the set of directed edges from CN to RN; for each cn∈CN, there may be 0∼n outgoing edges to nodes in RN, where n is a nonnegative integer, while there can only be 1 incoming edge for each rn∈RN. For each $e \in E_S$, $b(e)$ denotes the beginning of e and $t(e)$ denotes the terminus of e. We use repair(cn) to denote $\{rn \in RN | \exists e \in E_S, b(e) = cn \wedge t(e) = rn\}$.

(4) E_P is the set of directed edges from RN to CN; for each rn∈RN, there may be 0∼n outgoing edges to nodes in CN, and for each cn∈CN, there may be 0∼n incoming edges, where n is a nonnegative integer. For each $e \in E_P$, $b(e)$ denotes the beginning of e and $t(e)$ denotes the terminus of e. We use violation(rn) to denote $\{cn \in CN | \exists e \in E_P, b(e) = rn \wedge t(e) = cn\}$.

(5) Function h marks each node in CN and RN as a corresponding conjunction of literals.

(6) Function g marks each $e \in E_P$ as a pair of atoms $[g_{rn}, g_{cn}]$, where $g_{rn} \in h(b(e))$ and $g_{cn} \in h(t(e))$ are atoms that have the same predicate.

The algorithm for establishing the constraint dependency graph is given below.

For each *ED*, we set up a constraint node $cn \in CN$ denoting its left side. For each conjunction on the right side of the

TABLE 2. EDs corresponding to the axioms in the OWL RL ontology.

① $Teacher(t) \rightarrow TeacherName(t, n)$	⑩ $supervises(pt, ec) \rightarrow ExperimentCourse(ec)$
② $Course(c) \rightarrow CourseId(c, i)$	⑪ $Fulltime(ft) \rightarrow Teacher(ft)$
③ $Teacher(t) \wedge TeacherName(t, n_1) \wedge TeacherName(t, n_2) \wedge n_1 <> n_2 \rightarrow \bot$	⑫ $Parttime(pt) \rightarrow Teacher(pt)$
④ $Course(c) \wedge CourseId(c, i_1) \wedge CourseId(c, i_2) \wedge i_1 <> i_2 \rightarrow \bot$	⑬ $ExperimentCourse(ec) \rightarrow Course(ec)$
⑤ $TeacherName(t, n) \rightarrow Teacher(t)$	⑭ $supervises(pt, ec) \rightarrow teaches(pt, ec)$
⑥ $CourseId(c, i) \rightarrow Course(c)$	⑮ $Teacher(t) \rightarrow Fulltime(t) \vee Parttime(t)$
⑦ $teaches(t, c) \rightarrow Teacher(t)$	⑯ $Fulltime(t) \wedge Parttime(t) \rightarrow \bot$
⑧ $teaches(t, c) \rightarrow Course(c)$	⑰ $Teacher(t) \rightarrow Course(c) \wedge teaches(t, c)$
⑨ $supervises(pt, ec) \rightarrow Parttime(pt)$	⑱ $Course(c) \rightarrow Teacher(t) \wedge teaches(t, c)$

ED, we set up a repair node $rn \in RN$. Function h is applied to mark every cn and every rn, and the directed solid edges $e \in E_S$ from cn to rn are established. Since each *ED* has one and only one constraint node, we can identify each *ED* by its constraint node. If the repair node of some *ED* and the constraint node of another *ED* contain the same atom, we subsequently establish a directed dotted edge $e \in E_P$ from the former to the latter to record the repair/violation interaction between these two *EDs*; then, $e \in E_P$ is marked by function g. It is important to note that there may be more than one atom sharing the same pair of a repair node and constraint node, and the insertion of an object of each shared atom may result in a violation of the *ED* corresponding to the constraint node, so there may be multiple edges between one pair of a repair node and constraint node. Through algorithm 1, we can obtain the constraint dependency graph corresponding to the set of *EDs* listed in Tab. 2. In the interest of conciseness, we only give the correspondence between each *ED* and the set of *EDs* that have incoming edges from this *ED* (Tab. 3).

V. REDUCTION OF THE CONSTRAINT DEPENDENCY GRAPH

The next task is to delete as many *EDs* as possible while preserving the satisfiability result of $\mathcal{C}_{\mathcal{M}}$. First, we transform the first-order formula of $\mathcal{C}_{\mathcal{M}}$ into an *ED* and put it into the constraint dependency graph. Then, we iteratively execute operations to delete nondependent edges and *EDs* until no more operations can be executed; thus, we obtain the reduced graph. Taking the constraint dependency graph corresponding to Tab. 3 as an example, suppose that the target is the satisfiability of $Teacher \sqcap (\exists supervises.\ ExperimentCourse)$; we will obtain the result shown in Fig. 4. For the sake of clarity, we omit the information of functions h and g. Since there are only nine *EDs* and one of them is the target, the reduced graph is significantly smaller than the original one. This means that we only need to reason about the module with eight axioms.

Algorithm 1 Establishment of the Constraint Dependency Graph

Input: the set $\mathcal{D}$ of *EDs*;
Output: the constraint dependency graph $G =< CN, RN, E_S, E_P, h, g >$;
(1) Initialize $CN = \emptyset, RN = \emptyset, E_S = \emptyset, E_P = \emptyset$;
(2) For every $ED\ d \in \mathcal{D}$, do:
(3) Establish the constraint node cn of d, and let $CN = CN \cup \{cn\}$;
(4) Apply function h to mark cn;
(5) For every conjunction φ on the right side of d, do:
(6) Establish the repair node rn of φ, and let $RN = RN \cup \{rn\}$;
(7) Apply function h to mark rn;
(8) Establish edge e from cn to rn, and let $E_S = E_S \cup \{e\}$;
(9) Goto (5);
(10) Goto (2);
(11) For every $rn \in RN$ and every $cn \in CN$, do:
(12) For every atom $ar \in h(rn)$ and every atom $ac \in h(cn)$, do:
(13) If ar and ac have the same predicate, then:
(14) {Establish edge e from rn to cn, and let $E_P = E_P \cup \{e\}$
(15) Apply function g to mark e as $< ar, ac >$; }
(16) Goto (12);
(17) Goto (11);
(18) Return G.

A. TRANSFORMATION FROM $\mathcal{C}_{\mathcal{M}}$ TO ED

Now, we transform each $\mathcal{C}_{\mathcal{M}}$ into a denial constraint $\neg\varphi \rightarrow \bot$ by making the first-order formula φ of $\mathcal{C}_{\mathcal{M}}$ negative. This denial constraint, which indicates that it will be violated if $\mathcal{C}_{\mathcal{M}}$ is not satisfiable, is subsequently transformed into an *ED*, $\top \rightarrow \varphi$, through the procedure described in Section 3. For example, the target *Course* will be transformed into the

TABLE 3. Correspondence between the *EDs* in the constraint dependency graph.

ED i	*EDs* that have incoming edges from *ED i*	*ED i*	*EDs* that have incoming edges from *ED i*
①	③⑤	⑩	⑬
②	④⑥	⑪	①③⑮⑰
③		⑫	①③⑮⑰
④		⑬	②④⑱
⑤	①③⑮⑰	⑭	⑦⑧
⑥	②④⑱	⑮	⑪⑯ ⑫⑯
⑦	①③⑮⑰	⑯	
⑧	②④⑱	⑰	②④⑦⑧⑱
⑨	⑫⑯	⑱	①③⑦⑧⑮⑰

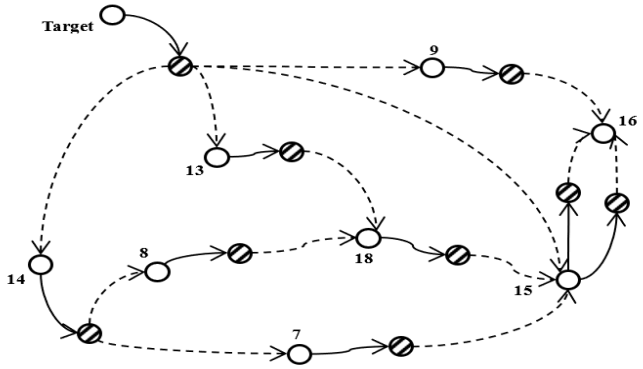


FIGURE 4. Reduced constraint dependency graph for the target ***Teacher*** ⊓ (∃***supervises.ExperimentCourse***).

$ED\ \top\ \rightarrow Course(c)$. Then, we put this new *ED* into the constraint dependency graph and record the repair/violation interaction between it and the other *EDs*, i.e., establish the edges $e \in E_P$ from the repair node of this new *ED* to the constraint nodes of the other *EDs*. Since the left side of $\mathcal{C}_{\mathcal{M}}$ *ED* is true, the corresponding constraint node does not have an incoming edge.

B. DELETING NONDEPENDENT EDGES

By analyzing the edges and nodes of G in depth, we find that the insertion of the beginning *rn* of some edges in E_P will never lead to the violation of their terminus *violation*(*rn*); thus, such edges can be deleted from G because they will not affect the results of the satisfiability check of $\mathcal{C}_{\mathcal{M}}$. We call such removable edges nondependent edges. Taking the repair node of *ED* 1 in Tab. 3 as an example, we see that this node points to the constraint nodes of *ED* 3 and *ED* 5. *ED* 1 and *ED* 3 state that each *Teacher* has at least and at most one name. In fact, the upper-bound cardinality constraint will never be violated by repairing the lower-bound cardinality constraint, because if *ED* 1 is violated, repairing it means that the name of the object of *Teacher* referred to by *ED* 1 does not exist at that time. Similarly, the repair of *ED* 1 will never lead to the violation of *ED* 5 because repairing *ED* 1 means that its left side *Teacher*(t) is true, while *Teacher*(t) is just the right side of *ED* 5; i.e., the right side of *ED* 5 is already true. Therefore, the two edges from *ED* 1 are both nondependent edges.

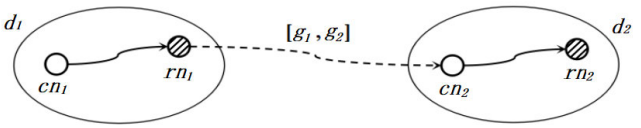


FIGURE 5. *EDs* for deleting nondependent edges.

Definition 2: (Nondependent Edge). *Given a constraint dependency graph G and an edge $e \in E_P$, e is called a nondependent edge if for an arbitrary objectification O, e satisfies one of the following conditions:*

(1) $O \cup \{h(b(e))\} \not\models h(t(e))$;

(2) $O \cup \{h(b(e))\} \models h(t(e))$, *and there exists a substitution* π *such that* $O \cup \{h(b(e))\} \models h(repair(t(e)))\pi$.

In the following passages, we provide strategies for deleting the nondependent edges. We describe these strategies by using the example shown in Fig. 5, which contains two *EDs*: d_1 and d_2. Given a repair node *rn* and the constraint node *cn* of *rn*, we use *increment*(*rn*) to represent the set of all the incremental variables in $h(rn)$, i.e., the set of all the variables that exist in $h(rn)$ but do not exist in $h(cn)$.

Theorem 1. *If there is a substitution* π *such that* $h(rn_2)\pi \subseteq h(cn_1) \cup h(rn_1)$, *where* $domain(\pi) \subseteq$ *increment*(rn_2), *then the edge* $[g_1, g_2]$ *is a nondependent edge.*

In short, whenever $h(cn_1)$ holds and we repair d_1 by inserting the object of $h(rn_1)$, $h(rn_2)$ always holds because it is already contained in $h(cn_1) \cup h(rn_1)$; so, d_2 will not be violated, although $h(cn_2)$ may be true as a result of the insertion of $h(rn_1)$. The relationship between *ED* 1 and *ED* 5 in Tab. 2 belongs to this case; hence, the edge between *ED* 1 and *ED* 5 is the nondependent edge that can be deleted.

Theorem 2. *If for every substitution* π *from the atoms of* $h(rn_1)$ *to the atoms of* $h(cn_2)$, *there is a substitution* π' *such that* $h(rn_1)\pi' \subseteq h(cn_2)\backslash h(rn_1)\pi$, *where* $domain(\pi') \subseteq$ *increment*(rn_1), *then edge* $[g_1, g_2]$ *is a nondependent edge.*

The relationship between *ED* 1 and *ED* 3 in Tab. 2 belongs to this case. If O' violates *ED* 3, then we delete the specific object of *TeacherName* that was just inserted for repairing *ED* 1 from the left side of *ED* 3. After deleting this object, we obtain the objectification O, which still contains one object of *TeacherName*; this object specifies the name of *Teacher*(t). Thus, we find that O satisfies *ED* 1, which is contradictory to the assumption.

Theorem 3. *If the domain of variable* $x \in increment(rn_1)$ *is infinite,* g_1 *contains* x*, and there is a predicate* p *in* $h(cn_2)$ *such that* $assignmentupper(h(rn_1), p, x) < assignmentlower(h(cn_2), p, x)$*, then edge* $[g_1, g_2]$ *is a non-dependent edge.*

In short, Theorem 3 describes such a case: since the repair of d_1 performed by inserting $h(rn_1)$ into O assigns a new value to x, d_2 will never be violated. Considering that there is an *ED* N whose left side contains $TeacherName(t_1, n) \wedge TeacherName(t_2, n)$, the repair of *ED* 1 will never lead to the violation of *ED* N, because during the repair of *ED* 1, the name assigned to *Teacher* is different from any existing value.

C. DELETING EDS

Now, we provide strategies for deleting *EDs*. Theorems 4 and 5 correspond to the first case described in Section 2.2, while Theorem 6 corresponds to the second case.

Theorem 4. *Given a constraint dependency graph G and a set* $\mathcal{D}$ *of the corresponding EDs of G, let cn be a constraint node such that* $h(cn) \neq \top$*; for an arbitrary constraint node* cn_i *such that* $h(cn_i) = \top$*, if there is no path from* cn_i *to cn, then the satisfiability of* $\mathcal{D}'$ *is equivalent to that of* $\mathcal{D}$*, where* $\mathcal{D}'$ *is the set obtained by deleting the ED of cn from* $\mathcal{D}$*.*

Theorem 5. *Given a constraint dependency graph G and the set* $\mathcal{D}$ *of the corresponding EDs of G, let p be a predicate that appears in* $h(cn)$*, where cn is a constraint node. If for every repair node* $rn \notin repair(cn)$*, p is not contained in* $h(rn)$*, then the satisfiability of* $\mathcal{D}'$ *is equivalent to that of* $\mathcal{D}$*,* where $\mathcal{D}'$ *is the set obtained by deleting the ED of cn from* $\mathcal{D}$*.*

Theorem 6. *Given a constraint dependency graph G and the set* $\mathcal{D}$ *of the corresponding EDs of G, for an arbitrary constraint node cn, if there exists a repair node* $rn \in repair(cn)$ *such that* $violation(rn)$ *is empty, then the satisfiability of* $\mathcal{D}'$ *is equivalent to that of* $\mathcal{D}$*,* where $\mathcal{D}'$ *is the set obtained by deleting the ED of cn from* $\mathcal{D}$*.*

After applying Theorems 1∼6 iteratively, we obtain the reduced constraint dependency graph shown in Fig. 4. Then, beginning with an empty ontology, we can obtain the satisfiability-preserving module of the original OWL RL ontology by adding the axiom corresponding to each *ED* in the reduced constraint graph to the new ontology. For the case of EquivalentClasses and EquivalentObjectProperties, we add these axioms to the new ontology if either of the two *EDs* exists in the reduced graph, because any individual *ED* is responsible for the checking result. For the graph in Fig. 4, the module obtained through the reconstruction process is shown in Fig. 2.

VI. EXPERIMENTAL EVALUATION

To verify the effectiveness of our method, we developed a prototype system and carried out many experiments. All the experiments were carried out on a platform with an Intel Core i5-7400, 4 GB RAM, and Windows 10. We chose three ontology repositories in the OWL 2 RL Benchmark Corpus [18]—ORB-Oxford Ontology, ORB-BioPortal and ORB-MOWLCorp—as the datasets. The OWL 2 RL Benchmark Corpus is an OWL RL repository extracted from general-purpose repositories, including the Oxford Ontology [19], BioPortal [20] and the Manchester OWL Corpus [21], by filtering out the content that does not fall under the OWL RL profile. In our tests, two reasoning engines, HermiT and Fact++, were used to carry out the satisfiability checking procedure.

For each repository, we randomly chose four concepts and checked their satisfiability. Hence, each concept and each reasoning engine combined to form a different target to be tested. The results of the checking time are shown in Figs. 6 ∼ 8. The detailed experimental results are summarized in Tabs. 4 ∼ 6.

For ORB-Oxford Ontology, we randomly chose four concepts: $\exists hasDisposition.Nervous$, *BusinessRelationship*, *GeneticInteraction* and $PhysicalRegion \sqcap \neg Airport$. The reasoning time comparisons for checking the satisfiability of these concepts are shown in Fig. 6. The legend represents the time needed for reasoning on the original ontology and extracted module. We can see that for both HermiT and Fact++, the reasoning time after the extraction is dramatically less than the original reasoning time. For ORB-BioPortal and ORB-MOWLCorp, we obtained results similar to those shown in Fig. 6. The experimental results summary for ORB-Oxford Ontology is shown in Tab. 4, which includes the number of *EDs* in the constraint dependency graph before and after the reduction, the time spent by the extraction process and the improvement in the checking efficiency due to the extraction. It is easy to see from Section 4 that the numbers of nodes and edges in the constraint dependency graph increase at a polynomial rate as the size of the original OWL RL ontology increases, and this is reflected in the average extraction time, which is only 8.50 s. The experimental results also show that the average checking time is reduced by 34.51 s (HermiT) and 67.26 s (Fact++) through the extraction. The summaries for the other two datasets are shown in Tabs. 5 and 6. The average checking time is reduced by 31.93 s (HermiT, ORB-BioPortal), 63.06 s (Fact++, ORB-BioPortal), 30.69 s (HermiT, ORB-MOWLCorp) and 42.02 s (Fact++, ORB-MOWLCorp), and the average time spent in the extraction process, by contrast, is only 8.14 s and 5.37 s.

VII. RELATED RESEARCH

The research related to our work concerns two tasks—module extraction and ontology partitioning—which are discussed below.

Module extraction has received much attention in recent years. Because extracting modules of minimal scale is known

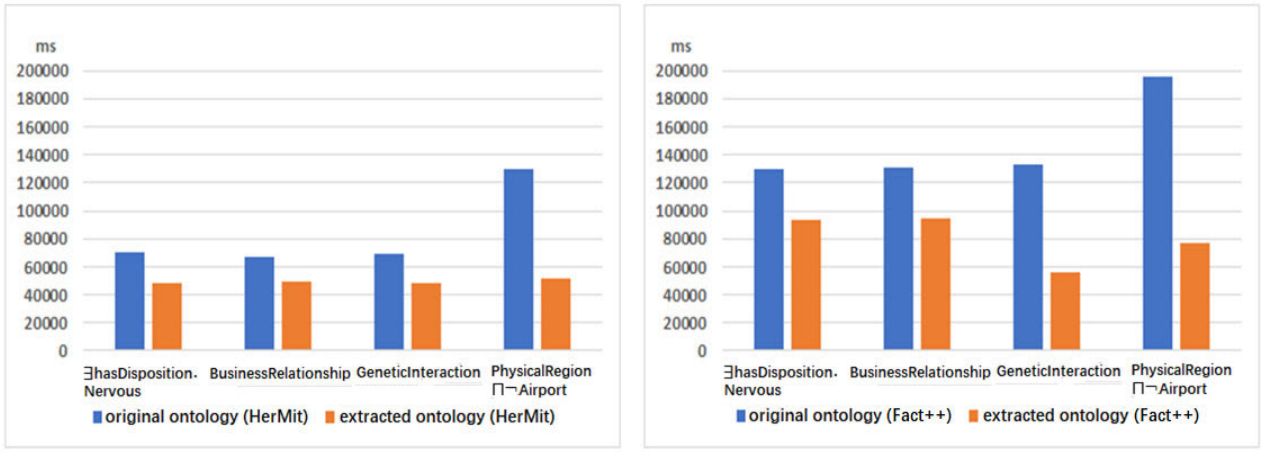


FIGURE 6. Checking times for ORB-Oxford Ontology concept satisfiability.

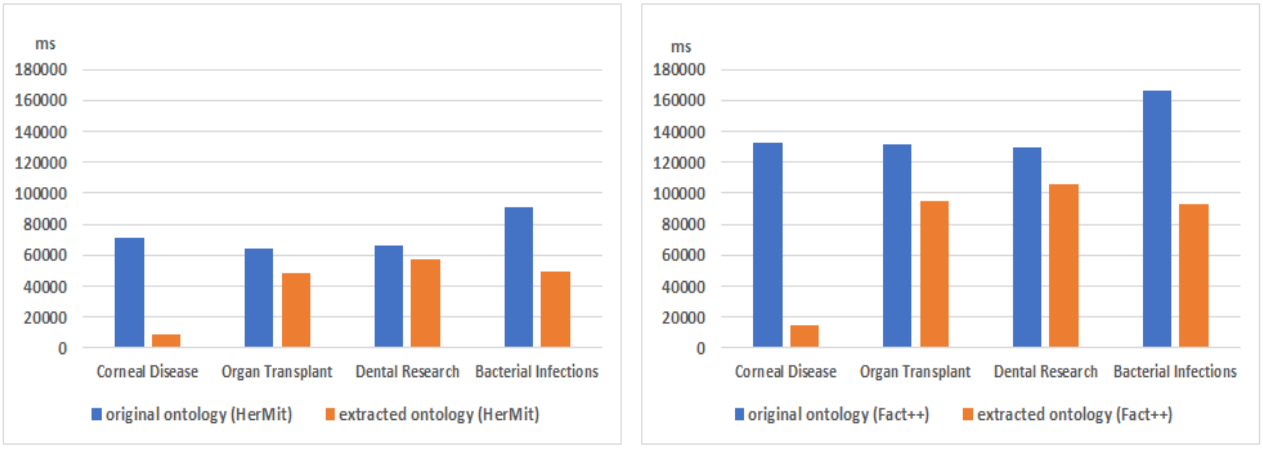


FIGURE 7. Checking times for ORB-BioPortal concept satisfiability.

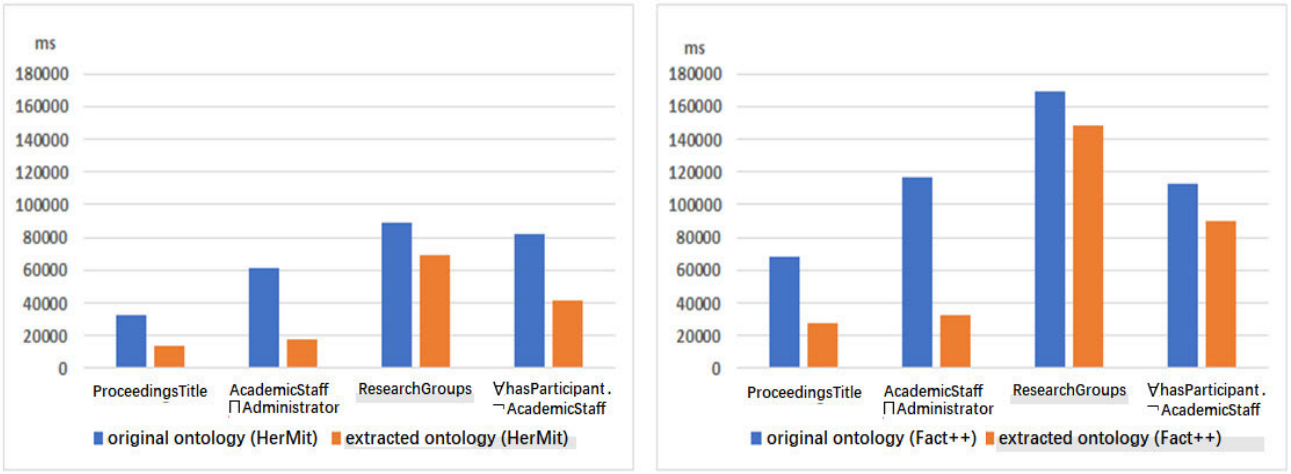


FIGURE 8. Checking times for ORB-MOWLCorp concept satisfiability.

to be computationally hard, especially for highly expressive ontology languages, practical techniques usually guarantee that the extracted module provably captures the relevant properties but do not guarantee that the module is minimal. Konev *et al.* [12] and Gatens *et al.* [13] investigate module extraction techniques for the description logics $\mathcal{ELI}$ and

TABLE 4. Experimental results for ORB-Oxford ontology.

	∃hasDisposition. Nervous	BusinessRelationship	GeneticInteraction	PhysicalRegion⊓ ¬Airport
Extraction time	9542 ms	8046 ms	7279 ms	9128 ms
ED (before reduction)	51	47	69	124
ED (after reduction)	16	18	27	39
HermiT time (original ontology)	69871 ms	66728 ms	69722 ms	129677 ms
HermiT time (extracted module)	48121 ms	49126 ms	48703 ms	52008 ms
Efficiency improvement	1.45	1.36	1.43	2.49
Fact++ time (original ontology)	129882 ms	130904 ms	133052 ms	195521 ms
Fact++ time (extracted module)	93019 ms	94328 ms	56180 ms	76788 ms
Efficiency improvement	1.40	1.39	2.37	2.55

TABLE 5. Experimental results for ORB-BioPortal.

	Corneal Disease	Organ Transplant	Dental Research	Bacterial Infections
Extraction time	8471 ms	8226 ms	7255 ms	8623 ms
ED (before reduction)	76	58	58	91
ED (after reduction)	21	39	42	64
HermiT time (original ontology)	70965 ms	64331 ms	66713 ms	91165 ms
HermiT time (extracted module)	8954 ms	48910 ms	57844 ms	49766 ms
Efficiency improvement	7.93	1.32	1.15	1.83
Fact++ time (original ontology)	132790 ms	131259 ms	129342 ms	166526 ms
Fact++ time (extracted module)	14629 ms	94506 ms	105880 ms	92661 ms
Efficiency improvement	9.08	1.39	1.22	1.80

$\mathcal{ALCQI}$, respectively. These techniques ensure that the modules preserve all entailments over a given vocabulary and that the modules are depleting and self-contained. They develop a polynomial algorithm for $\mathcal{ELI}$ and implement it in the system MEX. The algorithm for $\mathcal{ALCQI}$ is nontractable and is implemented in the system AMEX. To select the relevant axioms from $\mathcal{ELH}$ ontologies, Chen *et al.* [14] propose a novel module notion called the projection module that entails the queries that follow from a reference ontology. They develop an algorithm for obtaining the minimal projection module for subsumption and conjunctive queries. Georgia *et al.* [15] propose a method for extracting RDFS ontologies and implement it in the RDFDigest system. Based on the schema semantics, graphical structure and object distribution, they propose the notions of *relative cardinality* and *degree centrality,* which are used to describe the degree of association between the schema elements; thus, the candidate elements that constitute the module can be identified. Rakebul [16] proposes a method for extracting RDFS ontologies. The schema elements are divided into three categories: *salient statements*, *similar statements* and *abstract statements*. For each category, a set of filtering rules is defined so that a subset of the original schema can be obtained. By employing a notion of *level of detail* for modules in the deductive setting, Rousset and Ulliana [11] present a parametric semantics for bounded-level modules allowing to effectively control their size. Then, module extraction algorithms compliant with this novel semantics are provided and implemented on top of an RDF engine. Differently from our approach, all of these techniques focus exclusively on light-weight ontology languages, and do not permit to modularize Semantic Web datasets designed with rule-based implementations. The closest work to ours, at least in spirit, is [14]. However, it does not consider inference, which makes module extraction challenging.

TABLE 6. Experimental results for ORB-MOWLCorp.

	ProceedingsTitle	AcademicStaff⊓ Administrator	ResearchGroups	∀hasParticipant.¬AcademicStaff
Extraction time	5270 ms	4202 ms	5453 ms	6542 ms
ED (before reduction)	46	46	79	79
ED (after reduction)	11	14	61	36
HermiT time (original ontology)	32660 ms	61121 ms	89123 ms	82100 ms
HermiT time (extracted module)	13755 ms	17379 ms	69140 ms	41952 ms
Efficiency improvement	2.37	3.52	1.29	1.96
Fact++ time (original ontology)	67911 ms	116512 ms	169672 ms	112323 ms
Fact++ time (extracted module)	27698 ms	32772 ms	148011 ms	89840 ms
Efficiency improvement	2.45	3.56	1.15	1.25

Konev *et al.* [12] and Jongebloed and Schneider [22] investigate partition-based reasoning techniques in the context of description logics, where the goal is to improve the efficiency of reasoning over an ontology by first dividing its axioms into related partitions. The approach proposed by Georgia et al [23] focuses on RDFS ontologies, where the most important schema elements are selected as the cores, and then the remaining elements are assigned to the appropriate cores according to the dependencies among the data; thus, different partitions can be obtained. Although the output partitions are significantly smaller than the original ontology, their approach achieves high data redundancy, since the same data may be allocated to multiple partitions. A semantic-aware RDFS ontology partitioning method is presented in [24]. The schema elements are first sorted by using an improved page sorting algorithm, and then the cross-edges between different partitions are reduced according to their semantics. The key concern in partition-based reasoning is to find a partitioning that exhibits a suitable balance among the number of partitions, their size, and the number of common symbols among partitions to enable efficient distributed reasoning. In this setting, partitions are not necessarily property-preserving subsets of the input ontology and are therefore fundamentally different from the modules considered in our work.

VIII. CONCLUSION

To improve the efficiency of ontology reasoning, in this article, we propose a method for extracting modules from the OWL RL ontology. Given a concept to be checked, our method can return a subset that is significantly smaller than the original ontology without changing the satisfiability of the concept. We begin by formalizing the OWL RL axioms as a set of *EDs* and establishing a constraint dependency graph based on this set. Then, a series of operations are performed iteratively to delete the nondependent edges and the *EDs*. Finally, the module is obtained through a reconstruction process. For each reduction strategy, we give strict proofs. To show the effectiveness of our method, we developed a corresponding prototype system and carried out experiments that combine multiple datasets, reasoning engines and checking targets. The experimental results show that our method can significantly improve the checking efficiency. To the best of our knowledge, the presented work is the first implementation of module extraction from the OWL RL ontology.

State-of-the-art OWL RL reasoners, such as RuQAR and Arachne, currently do not rely on modules to split the workload. It would be natural to integrate our techniques into these tools for optimizing the reasoning. We believe that using our modules for reasoning will lead to a better use of these reasoners.

In the future, we also plan to investigate how our techniques could be exploited to improve the performance of query processing. Furthermore, we also envision potential applications to incremental reasoning, where data is frequently changing but queries and ontologies can be seen as fixed.

REFERENCES

[1] C. Golbreich and E. K. Wallace, "OWL2 Web ontology language new features and rationale," *W3C Recommendation*, vol. 11, pp. 1–8, Dec. 2012. [Online]. Available: https://www.w3.org/TR/2012/REC-owl2-new-features-20121211

[2] C. Roberto, S. Marco, and K. Oliver, "Conceptual blending in EL++," in *Proc. Int. Workshop Description Logics*, 2016, pp. 31–44.

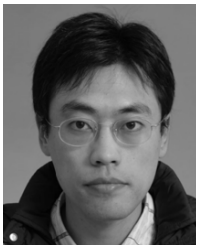
[3] C. Federico and L. Maurizio, "A framework for explaining Query answers in DL-lite," in *Proc. Eur. Workshop Knowl. Acquisition (EKAW)*, vol. 2018, pp. 83–97.

[4] D. Zheleznyakov, E. Kharlamov, W. Nutt, and D. Calvanese, "On expansion and contraction of DL-lite knowledge bases," *SSRN Electron. J.*, vol. 57, pp. 1–19, Dec. 2019.

[5] C. G. Bernardo, K. Evgeny, and V. K. Egor, "Controlled Query evaluation over OWL 2RL ontologies," in *Proc. 12th Int. Semant. Web Conf. (ISWC)*, 2013, pp. 49–65.

[6] B. Jarosaw and B. Michal, "RuQAR: Querying OWL 2RL ontologies with rule engines and relational databases," in *Proc. 9th Int. Conf. Comput. Collective Intell. (ICCCI)*, 2017, pp. 93–102.

[7] B. Glimm, I. Horrocks, B. Motik, G. Stoilos, and Z. Wang, "HermiT: An OWL 2 reasoner," *J. Automated Reasoning*, vol. 53, no. 3, pp. 245–269, Oct. 2014.
[8] T. Dmitry, "Incremental and persistent reasoning in FaCT++," in *Proc. 3rd Int. Workshop OWL Reasoner Eval.*, 2014, pp. 69–75.
[9] A. A. Romero, M. Kaminski, C. G. Bernardo, and I. Horrocks, "Ontology module extraction via datalog reasoning," in *Proc. 29th AAAI Conf. Artif. Intell. (AAAI)*, 2015, pp. 1410–1416.
[10] A. Armas Romero, M. Kaminski, B. Cuenca Grau, and I. Horrocks, "Module extraction in expressive ontology languages via datalog reasoning," *J. Artif. Intell. Res.*, vol. 55, pp. 499–564, Feb. 2016.
[11] M. C. Rousset and F. Ulliana, "Extracting bounded-level modules from deductive RDF triplestores," in *Proc. 29th AAAI Conf. Artif. Intell. (AAAI)*, 2015, pp. 268–274.
[12] B. Konev, C. Lutz, D. Walther, and F. Wolter, "Model-theoretic inseparability and modularity of description logic ontologies," *Artif. Intell.*, vol. 203, Oct. 2013, pp. 66–103.
[13] W. Gatens, B. Konev, and F. Wolter, "Lower and upper approximations for depleting modules of description logic ontologies," in *Proc. 21st Eur. Conf. Artif. Intell. (ECAI)*, 2014, pp. 345–350.
[14] J. Chen, M. Ludwig, Y. Ma, and D. Walther, "Computing minimal projection modules for ELH-terminologies," in *Proc. 16th Eur. Conf. Logics Artif. Intell.*, 2019, pp. 355–370.
[15] T. Georgia, K. Haridimos, D. Evangelia, and P. Dimitris, "RDF Digest: Efficient summarization of RDFS KBs," in *Proc. 12th Eur. Semant. Web Conf. (ESWC)*, vol. 2015, pp. 119–134.
[16] H. Rakebul, "Generating and summarizing explanations for linked data," in *Proc. 11th Eur. Semant. Web Conf. (ESWC)*, 2014, pp. 473–487.
[17] B. Motik, B. C. Grau, and I. Horrocks. (Nov. 2012). *OWL 2 Web Ontology Language profiles*. [Online]. Available: https://www.w3.org/TR/owl2-profiles/#Reasoning_in_OWL_2_RL_and_RDF_Graphs_using_Rules
[18] *OWL 2 RL Benchmark Corpus*. Accessed: Dec. 21, 2019. [Online]. Available: http://mowlrepo.cs.manchester.ac.uk/datasets/owl-2-rl-benchmark-corpus
[19] *Oxford Ontology Repository*. Accessed: Apr. 11, 2020. [Online]. Available: http://www.cs.ox.ac.uk/isg/ontologies
[20] *The National Center for Biomedical Ontology*. Accessed: Jan. 30, 2020. [Online]. Available: http://bioportal.bioontology.org
[21] *Manchester OWL Repository*. Accessed: Mar. 28, 2020. [Online]. Available: http://mowlrepo.cs.manchester.ac.uk/datasets/mowlcorp
[22] S. Jongebloed and T. Schneider, "Ontology partitioning using *E*-connections revisited," in *Proc. 31st Int. Workshop Description Logics*, 2018, pp. 161–174.
[23] T. Georgia, K. Haridimos, and P. Dimitris, "Semantic partitioning for RDF datasets," in *Proc. 11th Int. Workshop Inform. Search, Integr., Personalization (ISIP)*, vol. 2016, pp. 99–115.
[24] Q. Xu, "Semantic-aware partitioning on RDF graphs," in *Proc. 1st Int. Joint Conf. Asia-Pac. Web Web-Age Inform. Manag. (APWeB-WAIM)*, vol. 2017, pp. 149–157.
[25] T. Özacar, Ö. Öztérk, and M. O. Ünalèr, "ANEMONE: An environment for modular ontology development," *Data Knowl. Eng.*, vol. 70, no. 6, pp. 504–526, Jun. 2011.
[26] C. Shimizu, A. Krisnadhi, and P. Hitzler, "Modular ontology modeling: A tutorial," *Appl. Practices Ontol. Des., Extr., Reasoning*, vol. 11, pp. 54–73, Aug. 2020.
[27] W. Terkaj and P. Pauwels, "A method to generate a modular ifcOWL ontology," in *Proc. 8th Int. Workshop Form. Ontol. Meet Ind.*, 2017, pp. 78–89.
[28] B. Cuenca Grau, I. Horrocks, Y. Kazakov, and U. Sattler, "Modular reuse of ontologies: Theory and practice," *J. Artif. Intell. Res.*, vol. 31, pp. 273–318, Feb. 2008.
[29] D. Lonsdale and D. W. Embley, "Reusing ontologies and language components for ontology generation," *Data Knowl. Eng.*, vol. 69, no. 4, pp. 318–330, 2010.
[30] E. Jimenez-Ruiz and B. C. Grau, "Logmap: Logic-based and scalable ontology matching," in *Proc. Int. Semant. Web Conf. (ISWC)*, 2011, pp. 273–288.
[31] M. Ludwig, "Just: A tool for computing justifications wrt ELH ontologies," in *Proc. 3rd Int. Workshop OWL Reasoner Eval. (ORE)*, 2014, pp. 1–7.
[32] K. Patrick and R. A. Schmidt, "Count and forget: Uniform interpolation of SHQ-ontologies," in *Proc. 7th Int. Joint Conf. Autom. Reasoning (IJCAR)*, vol. 2014, pp. 434–448.
[33] K. Patrick, "LETHE: Forgetting and uniform interpolation for expressive description logics," *Kunstliche Intelligenz*, vol. 34, no. 1, pp. 866–872, 2020.
[34] A. A. Romero, B. C. Grau, and I. Horrocks, "MORe: Modular combination of OWL reasoners for ontology classification," in *Proc. Int. Semant. Web Conf. (ISWC)*, vol. 2012, pp. 1–16.



XIAOFEI ZHAO received the M.S. degree in computer software and theory from the Dalian University of Technology, Dalian, China, in 2004, and the Ph.D. degree in computer application technology from the Nanjing University of Aeronautics and Astronautics, Nanjing, China, in 2008.

From 2011 to 2013, he held a postdoctoral position at the Institute of Computing Technology, Chinese Academy of Sciences. From 2013 to 2015, he was a Lecturer with the School of Computer Science and Technology, Tiangong University, Tianjin, where he has been an Associate Professor since 2015. His research interests include knowledge engineering and web engineering.

FANZHANG LI received the M.S. degree in computer science and technology from the University of Science and Technology of China, Hefei, China, in 1995.

From 1997 to 1999, he was an Associate Professor with the School of Information Science and Engineering, Yunnan University, Kunming, where he has been a Professor since 1999. Since 2000, he has been working with the School of Computer Science and Technology, Soochow University, Suzhou. His main research interests include knowledge representation and reasoning, and dynamic fuzzy logic.

Prof. Li's awards and honors include the IEEE CS GRC Pioneer Award and the Provincial Science and Technology Progress Award.

HONGJI YANG (Member, IEEE) received the B.S. and M.S. degrees in computer science from Jilin University, Changchun, China, in 1982 and 1985, respectively, and the Ph.D. degree in computer science from Durham University, Durham, U.K., in 1994.

From 2003 to 2013, he was a Professor with the Department of Computer Technology, De Montfort University. From 2013 to 2017, he was a Professor with Bath Spa University. Since 2018, he has been working with the School of Informatics, University of Leicester. His main research interests include knowledge modeling and creative computing. He became a Golden Core Member of the IEEE Computer Society in 2010.