A METHOD FOR IMPROVING REASONING AND REALIZATION PROBLEM SOLVING IN DESCRIPTIVE LOGIC- BASED AND ONTOLOGY-BASED REASONERS

Mojtaba Shokohinia[1], Abbas Dideban[2*], Farzin Yaghmaee [3]

[1,2,3]Department of Electrical and Computer Engineering, Semnan University, Semnan, Iran

E-mail: mshokohinia@semnan.ac.ir [1], adideban@semnan.ac.ir [2*] (corresponding author), f_yaghmaee@semnan.ac.ir [3]

DOI: https://doi.org/10.22452/mjcs.vol35no1.3

ABSTRACT

Recently, many methods have been developed for representing knowledge, reasoning, and result extraction extracting results based on the respective domain knowledge in question. Despite the ontological success in knowledge representation, the reasoning method has faces some challenges. The main challenge in ontology reasoning methods is the failure in solving realization problems in the reasoning process. Apart from the complexity of solving realization problems, this already daunting challenge is compounded by computational complexity the time complexity of the solving realization problem solving process problems is equal to that of NEXP TIME. This important issue problem is achieved solved by solving the subsumption and satisfiability problems. Thus, to solve the realization problem, we first partition the ontology or extract partitions related to the query. Then, the satisfiability problem is solved by extracting partitions, and all concepts related to the query are extracted. This study proposes a method to overcome this problem, where a new solution is proposed with an appropriate time position. Finally, the efficiency of the proposed method, is evaluated against other reasoning engines, and the results show optimized performance vis-a-vis previous studies.

Keywords: Semantic Web, Reasoning, Descriptive Logic, Ontology, Subsumption

1.0 INTRODUCTION

Data are mainly used in real-world and laboratory applications. However, due to the special aspects arising from the nature of the data, challenges also arise. One such challenge, one of which is the efficient and careful processing of real-world data or data analysis for reasoning and extraction of its rules and inherent logic. When the data are semantic, this problem becomes even more difficult. Thus, the reasoning type must be semantic and correspond to the data. In recent years, scholars have conducted done considerable research in the field of reasoning, especially semantic reasoning, to accelerate this area of research. The main goal of research in the field of reasoning is to improve machine understanding in the automatic process of reasoning. Generally, different types of reasoning, such as abductive reasoning, backward reasoning, and comparative reasoning, are used in their specific fields and applied to automatic and machine reasoning during the year. It describes a type of reasoning in which the hypothesis generation process is a description of the data [1]. In this study, abductive reasoning with ontological use is presented. First, the use of this type of reasoning in ontology is discussed. Then, the function of this type of reasoning in ontology is examined by defining some scenarios. Moreover, reasoning is the main problem in OWL-DL, which requires high computational cost. The integration of rules makes reasoning indecisive. In addition to the above limitations, ontology reasoning in OWL-DL has many practical limitations. In other words, the normal solution to obtain rich data using ontological reasoning is to solve the realization problem. Realization in OWL-DL is about finding the most specific classes to which an individual belongs; in other words, it is about computing the direct types for each individual. Realization is about finding the most relevant class of objects. However, realization is a problem with time complexity equivalent to that of NEXP TIME. The online performance of ontological reasoning has fundamental scalability and execution time issues, especially when the ontology has many individuals [2]. So far, various optimizations have been presented by relational database techniques to improve the reasoning performance of OWL-DL. Instance store is one of the most popular methods [3], which also has crucial limitations in reasoning over RDF data.

Recently, many methods have been developed for representing knowledge, reasoning, and extraction results. Despite the ontological success in knowledge representation, the reasoning method faces some challenges. The lack of problem realization in the reasoning process is the most important challenge in ontology reasoning. Apart from the complexity of solving realization problems, this already daunting challenge is compounded by computational complexity. The time complexity of realizing and solving the problem is equal to NEXP time. In this study, an optimal solution with reasonable time complexity is proposed. In this study, the realization problem has been divided into its concepts and smaller sub-problems and solved to improve the ontology reasoning process. The main goal is to overcome the realization complexity. This is done by solving the subsumption problem and the satisfiability problem. To solve the realization problem, the ontology is first partitioned, or partitions are related to the query. Then, the satisfiability problem by extracting partitions, and extracting all concepts related to the query are extracted. Next, the subsumption is solved for each identified pair of concepts in the satisfiability problem is solved until the end of the presented algorithm. Concepts that are not included in each of the subsets are considered as answers to the query. The following two issues need to must be considered.

First, the concept of the query is the solution to the realization problem of individual A. For this concept, the most specific concept with respect to individual A is identified and then extracted. Second, at the end of the presented algorithm, the same concepts that are candidates for the most specific concept are obtained, whose sum is the final answer to the query. Thus, the new concepts are the best answer to the realization problem. In rare cases, it may also be an empty set where the sum of several empty sets is empty. The results show that this method works accurately. Thus, more time is saved and a faster answer is obtained. Moreover, the ontology partition shows that the proposed method achieves higher optimal proficiency than the existing methods. In the continuation of this article, the second part is dedicated to the literature review. The third part introduces descriptive logic and reasoning as basic concepts. The fourth part presents the methodology. The fifth part also provides a relevant example. The sixth part analyzes the results and studies the accuracy and execution time using the reasoning engines HermiT[4] and FaCT++ [5]. The results show that the proposed method outperforms the other methods. Finally, the seventh section is dedicated to the conclusions of the study.

2.0 LITERATURE REVIEW

This section describes the work done in this regard. A reasonable and thorough proof method for backward compatibility reasoning is presented, as well as a fully abstract trace-based semantics for class libraries [6]. The authors of [7] suggest a modular reasoning system that reuses code by using uninterpreted predicates. A collection of publications on the history of medical sciences to find recurrent patterns of queries is examined in [8]. According to the suggested paradigm, the queries are represented in SQWRL, which provides a mechanism for applying temporal reasoners to answer inquiries about the history of science. Comparative reasoning models for various applications, such as RDF data where an agent is compared to other agents, are investigated by [9], [10]. A distributed reasoner with excellent efficiency in [11] is expressed. Its main limitation is that it only supports classification. More precisely, it is a distributed ontology classifier. According to [12], this reasoner is a meta-reasoner that uses a set of reasoners for inference, with its main drawback being its operation and query structure writing, which have made it unpopular. A free and open-source reasoner is introduced in [13]. Although this lightweight reasoner is highly efficient, it does not support many of the capabilities of a reasoner. The authors of [14] present the quality of crowdsourced relevance judgments regarding the ability to reason. This reasoner uses a forward-chaining inference engine whose efficiency is high and whose main problem is that it does not support the classification problem. A special model for inductive reasoning for searches and queries in the semantic web is presented in [15]. The main problem of this reasoner is its non-scalability, reducing its efficiency for large data sets. This model starts with detailed and specific rules and ends with general rules. The authors of [16] describe an automated user interface based on reasoning, a free and open-source Java programming language. This reasoner is an OWL 2 DL reasoner with excellent efficiency on small data sets. However, its efficiency decreases significantly as the volume of the ontology increases. One of the challenges with a search engine that considers semantic information is that it requires index information beyond the stored traditional information, including entity mentions and type relationships [17]. Moreover, the performances of different and the same data structure types are discussed, the different indexes are described, and the appropriate index type for semantic search under different conditions is determined. In [18], query ambiguity resolution by query expansion

using a statistical linguistic technique during semantic query formulation is described to retrieve relevant answers from a Holy Quran ontology. Finally, [19] presents a black-box approach to parallelizing current description logic (DL) reasoners for the Web Ontology Language (OWL), resulting in a parallel framework that can be used to existing OWL reasoners to speed up their categorization process.

Backward reasoning is another type of reasoning where the intended goal of the reasoning process is approached backward in order to examine the effective factors in achieving particular goals. This method is particularly in line with optimization. From another point of view, semantic reasoning can be divided into four classes. The solutions for semantic data reasoning model can also be divided into four main categories as follows:

The first group has extended the display of semantic data to add rules and facts to evaluate the capability of reasoning rules. The main shortcoming of these methods is that with the modification and extension of semantic data to improve reasoning ability and define rules and facts, the data usually lose their essential and natural structure as well as their main semantic features [20], [21], [22]. The second group includes usage-based methods. In other words, in these methods, the data are transformed into an ontology structure, and the semantic data are used in the common reasoning model of these data to improve the quality of usage. For example, in an IP TV recommender system, semantic data and current ontologies are used to infer the appropriate recommendations for each user [23], [24]. The third group includes the methods in which the data model and the type of reasoning in semantic data are first converted into a logic model, such as first-order logic (FOL) or other logic types, and then one of the reasoning mode in logic is used to conduct reasoning on semantic data [25], [26], [27].

The fourth group is the same as the third group, with only one difference. That is, in addition to converting the semantic data structure into temporal logic, the semantic data themselves are converted to the temporal data structure, and reasoning is performed by the middle logic on temporal data [28], [29]. According to [28], [29]. Compared with other simpler, informal, and innovative methods, ontological methods not only have more advantages in their simplicity but also in their integration. The development Ontology Web Language-Descriptive Logic (OWL-DL)-based ontology models shows that this technique is sometimes inefficient in defining and understanding complex relationships and descriptions. Therefore, the constructors in the OWL-DL language are chosen to support the decidable reasoning process. Any process beyond that has no possibility of reasoning in OWL-DL [30]. Therefore, OWL-DL does not include explicit constructors, such as user activities, which may be useful for modelling complex domains. Recently, the semantic web community has conducted studies on the possible extension of OWL-DL features, the results of which are logical languages, such as SWRL, as used in [31].

See Table 1 for a list of reasoners and their supporting features. One of the main features of the reasoner is realization, which is the most complex problem in DL. The authors introduce the concept of bundle in [32]. However, as it uses probabilistic and palette-based contexts, it requires a high computational cost and a long runtime, which is not efficient for large data sets. [33] States one of the first reasoners for descriptive logic. The main drawback of this reasoner is that it is used to solve subsumption problems and is not efficient for classification problems. [34] Introduces Clipper, a reasoner for conjunctive query answering based on queries. The main drawback of this reasoner is its high time complexity and inefficiency on large data sets. DBOWL reasoner is presented in [35], which is an OWL reasoner whose main advantage is its scalability. An important disadvantage of this reasoner is that it has a precomputation step, and the data must be precomputed before being input to the reasoner to convert it to the correct format for the reasoner. DeLorean describes a fuzzy reasoner that implements fuzzy logic well [36]. The main limitation of this reasoner is that the input data must be fuzzy, and it does not support non-fuzzy data. Another drawback of this reasoner is its high time complexity. DRAGON [37] is an OWL reasoner used in a distributed manner. The main disadvantage of this reasoner is that it does not support the classification problem and, therefore, cannot be used for some applications. The authors of [38] present the a reasoner based on conjunctive query answering for DL programs. It performs well in supporting descriptive logic, but its time complexity is very high, making it inefficient for real-world applications. The jCEL reasoner is a rule-based reasoner designed for descriptive logic. This reasoner uses a rule-based completion algorithm for inference. Its time and memory overhead are substantial (high time and space complexity) [39]. The reasoner introduced in [40] is very efficient in terms of time complexity. An important property of this reasoner is that it can reason in parallel until the inference is significantly reduced. However, its main drawback is its inefficiency in solving classification problems, rendering it useless for some applications. [41] states LiFR (Lightweight Fuzzy semantic Reasoner), which is a useful reasoner based on fuzzy logic. Since it is a first version and lightweight, its efficiency is good; however, it does not support much of the descriptive logic. [42] is a prototypical reasoner used to solve classification problems. Its performance in terms of time complexity is not reasonable, and it

does not support other descriptive logic problems. As can be seen in Table 1, only a few reasoners support the realization function. Against this background, this study aims to solve the realization problem efficiently, and the details are discussed in the following part. It proposes an efficient method for solving the realization problem based on satisfiability and subsumption problems. Computational complexity is the main problem in solving the realization problem. The proposed method manages to adequately address this drawback to some extent. The proposed method is implemented in a standard reasoner and is compared with other reasoners in a standard framework. The results revealed an optimized performance compared to previous studies. The criteria for assessment of the introduced reasoners in this framework are realization, classification and consistency.

3.0 DESCRIPTIVE LOGIC AND REASONING

DLs belong to the formal family of knowledge representation. Many DLs are more accurate than propositional logic and less accurate than FOL. The main problems of DLs are decidable such that a logical and competent procedure has been developed for them. DLs are used in artificial intelligence to describe and reasoning about the relationships between concepts in the application domain. DLs provide a formal and logical definition for ontologies and a semantic web. General, DLs model concepts, roles, individuals, and correlations thereof. An axiom (i.e., a logical statement related to the roles and/or concepts) is modelled in the most basic concept in DLs. Ontology is the precise and formal description of the features of the context of study. In addition, it defines a common word for all researchers who want to share information about a particular domain. In general, ontology languages presented for the semantic web are a syntactic variant of DLs [43]. In DLs, there are two different meanings of Terminological Box (TBox) and Assertional Box (ABox). In a general sense, TBox contains sentences describing concept hierarchies (i.e., the relationship between concepts), while ABox contains sentences indicating where in the hierarchy individuals belong (i.e., the relationship between individuals and concepts). For example: “"Every employee is a person,”" is related to TBox.
“"Bob is an employee,”" is related to ABox.

The FOL syntax is defined as a series of legal symbols in DL. In contrast to FOL, DL has some known syntactic variants. Different operators are defined by recursive definitions in the DL family. A group of these operators, such as intersection or conjunction of concepts, union or disjunction of concepts, negation or complement of concepts, and universal and existential restrictions, are presented in FOL. Other operators, such as inverse, transitivity, and functionality involving operators with limitations on roles are related to FOL. A list of operators and DL symbols are presented in Table 2. In addition to showing the formal relationships among concepts, DL must be able to answer the questions about certain concepts. Database-query-likes are the most commonly proposed queries, such as:

1. Instance checking [is a particular instance (member of ABox and member of a particular concept)].
2. Relation checking (includes a relationship/role between two instances; in other words, A has the same attributes as B).
3. Subsumption (is a concept that is a subset of another concept).
4. Concept consistency (there is no contradiction among the definitions or chain of definitions).

In addition to the reasoners presented in Table 3, reasoning can also be used in various search engines. In other words, semantic search engines use reasoning on semantic data and RDF data when answering the related query. The following tables, provide a list of semantic search engines that use reasoning, and their reasoning types.

JESS [52] is an SWRL-enabled rule-based reasoner like PELLET [49], KAON2 [53], HOOLET [3], SHER [54]. The rule-based reasoner employs "If-Then-Else" statements and has two reasoning strategies: forward and backward chaining. The main challenge for rule-based reasoners is to change the number or all of the rules in the inference every time a change is made. These challenges make changes and maintenance difficult for this type of reasoner. The table below lists some other reasoners as variations of rule-based reasoners whose basis is the "If-Then-Else" statements. Some other reasoners like FACT [55] and FACT ++ [56] are not rule-based. They are tableaux-based reasoners for expressive description logics covering OWL and OWL2. This type of reasoners is based on formal logic and is more descriptive like rule-based reasoners. Therefore, description-logic-based reasoners are very complicated to design and implement. Nevertheless, they are very descriptive and can be easily adapted to reasoning strategies since the main concepts of description logic are applied to the reasoning algorithm.

Table 1: List of Descriptive logic Reasoners with their supported services

Reasoner	Institution	License	Details	Supported Reasoning Services					
				Satisfiability	Entailment	Consistency	Explanation	Realization	OWL
BUNDLE [32]	University of Ferrara	AGPL License version 2.0	Probabilistic reasoner based on Pellet	Yes	Yes	Yes	Yes	No	Yes
CEL [33]	Technische Universität Dresden	Apache License 2.0 (CEL) / GNU Lesser General Public License 3.0	Lisp-based reasoner	Yes	No	Yes	No	No	Yes
Clipper [34]	Vienna University of Technology	Apache 2.0	Reasoner for conjunctive query answering over Horn-SHIQ ontology via query rewriting	No	No	No	No	No	Yes
DBOWL [35]	University of Malaga	GNU General Public License	scalable reasoner for OWL ontologies with very large Aboxes	Yes	No	Yes	No	No	No
DeLorean [36]	Not given	Published under NA	Fuzzy rough Description Logic reasoner	Yes	Yes	Yes	Yes	Yes	Yes
DRAGON [37]	University of Paris 8, IUT of Montreuil	LGPL	OWL reasoner that supports distributed reasoning over a networked ontologies	No	Yes	Yes	No	No	Yes
DReW [38]	Vienna University of Technology	Apache 2.0	DReW is a reasoner for DL-Programs over Datalog-rewritable Description Logics for Conjunctive query Answering	No	No	No	No	No	Yes
Jcel [39]	Technische Universität Dresden	Apache License 2.0 and GNU Lesser General Public License 3.0	Free open-source Java-based reasoner for EL+ and supports parts of the OWL 2 EL profile	Yes	Yes	Yes	No	No	Yes
Konclude [40]	University of Ulm, derivo GmbH	LGPL 2.1	Parallel, high-performance reasoner for the Description Logic SROIQV(D)	Yes	Yes	Yes	No	Yes	No
LiFR [41]	Centre for Research and Technology Hellas (CERTH)	GNU LGPL	Lightweight Fuzzy DL Reasoner, capable of performing in resource-constrained devices	Yes	Yes	Yes	No	No	No
MORe [42]	University of Oxford	GNU Lesser GPL	MORe uses module extraction techniques to classify ontologies combining reasoners especially optimised for different OWL 2 profiles	Yes	No	No	No	No	Yes
ELK [44]	Not given	GNU GPL v3	Reasoner for log-linear description logics, a probabilistic logical formalisms that combines description logics and log-linear models	No	Yes	No	No	Yes	Yes
fuzzyDL [45]	ISTI – CNR	Published under NA	Free Java/C++ based reasoner for fuzzy SHIF with concrete fuzzy concepts	Yes	Yes	Yes	No	No	Yes
DistEL [46]	Wright State University	Published under NA	Distributed reasoner that runs on a cluster of machines	No	No	No	No	Yes	No
Chainsaw [47]	University of Manchester	LGPL	OWL 2 DL reasoner for very large ontologies	Yes	Yes	Yes	No	Yes	Yes
BaseVISor [48]	VIStology Inc.	Academic and research use free of charge	Forward-Chaining inference engine based on Rette	Yes	Yes	Yes	Yes	Yes	Yes
Pellet [49]	Clark & Parsia, LLC	AGPL v3	Free open-source Java-based reasoner for OWL 2 and SWRL	Yes	Yes	Yes	Yes	Yes	Yes
JFact [50]	University of Manchester	LGPL	pure Java port of FaCT++, with versions for Owlapi 3.x and 4.x	Yes	Yes	Yes	No	Yes	No
ELepHant [51]	Not given	Apache License, Version 2.0	Consequence-based reasoner that currently supports part of the OWL 2 EL fragment for the reasoning tasks classification, consistency and realization.	No	No	Yes	No	Yes	Yes
FaCT++ [5]	University of Manchester	LGPL	Free (LGPL) highly optimised open-source C++-based tableaux reasoner for OWL 2 DL	Yes	Yes	Yes	Yes	Yes	Yes
HermiT [4]	University of Oxford	LGPL	OWL 2 DL reasoner	Yes	Yes	Yes	Yes	Yes	Yes

Table 2: List of operators and DL symbols

Symbol	Description	Example	Read
⊤	⊤ is a special concept with every individual as an instance	⊤	Top
⊥	Empty concept	⊥	Bottom
⊓	Intersection or conjunction of concepts	$C \sqcap D$	***C*** and ***D***
⊔	Union or disjunction of concepts	$C \sqcup D$	***C*** or ***D***
¬	Negation or complement of concepts	$\neg C$	Not ***C***
∀	Universal restriction	$\forall R.C$	All ***R***-successors are in ***C***
∃	Existential restriction	$\exists R.C$	An ***R***-successor exists in ***C***
⊑	Concept inclusion	$C \sqsubseteq D$	All ***C*** are ***D***
≡	Concept equivalence	$C \equiv D$	***C*** is equivalent to ***D***
≐	Concept definition	$C \doteq D$	***C*** is defined to be equal to ***D***
:	Concept assertion	$a:C$	***a*** is a ***C***
:	Role assertion	$(a,b):R$	***a*** is ***R***-related to ***b***

Table 3: List of search engines with reasoners

Search Engines	OWL-DL Entailment	Supported Expressivity For Reasoning	Reasoning Algorithm	Rule Support
PELLET [49]	Yes	SROIQ(D)	Tableau	Yes (SWRL-DL Safe Rules)
JESS [52]	Yes	-	Rule – Based	Yes (SWRL)
KAON2 [53]	Yes	SHIQ(D)	Reasoning & Datalog	Yes (SWRL-DL Safe Rules)
HOOLET [3]	Yes	-	First-|Order Prover	Yes (SWRL)
SHER [54]	Yes	SHN	Rule – Based	Yes (SWRL-DL Safe Rules)
FACT [55]	Yes	SHIQ	Tableau	No
FACT++ [56]	Yes	SROIQ(D)	Tableau	No
RACERPRO [57]	Yes	-	Tableau	Yes (SWRL-not fully support SWRL)
JENA [58]	Yes	Varios Reasoner (incomplete for nontrivial description logic)	Rule - Based	Yes (own Rule Format)
F-OWL [59]	Yes	-	Tableau	Yes (own Rule Format)
SWEETRULES [60]	No	-	Rule - Base	Yes (SWRL, RuleML, Jess)
OWLIM [61]	No	R-entailment	Rule - Based	Yes (own Rule Format)
BASEVISOR [62]	No	R-entailment	Rule - Based	Yes (SWRL, RuleML, Jess)

4.0 METHODOLOGY

As mentioned in the previous part, ontology reasoning has significant limitations in its realization; therefore, this study aims to find a solution to improve it. The main limitation of ontology reasoning is its inability to solve the realization problem. Until now, various solutions have been used to overcome this inability.

The approach used in this study is to logically separate the realization problem. In other words, instead of choosing a particular subcategory of ontology, this study finds an answer to the realization problem by dividing it into some subcategories and then solving these subcategories. In this way, computational cost and inefficiency are avoided by minimizing the problem. In general, the advantages of this innovation can be summarized as follows:

1. Solving the realization problem in semantic reasoning.
2. Break up the realization problem to subsumption and satisfiability problems.
3. Solving the realization problem in Polynomial time.
4. Reasoning time is improved and the faster response is achieved.
5. Better precision and proficiency vis-a-vis the existing methods

In general, the realization problem can be divided into the following subcategories:

* Satisfiability of the concept: diagnosis of the concept considering the individual's affiliation based on their description.
* Subsumption of the concept: determining whether d concept d follows c, which means that c is more general than d.

In contrast, the realization problem can be defined as “"finding a concept considering where an individual has the most attachment.." To solve either of the above two problems, it is necessary to separate them and integrate their results.

Based on the definition of DL, each DL-based reasoning system consists of a TBox and an ABox, which we have already discussed.

Then, having a common TBox and ABox is necessary to solve and integrate the two problems [63]. However, TBox possibly expands from one problem to another, but it does not harm the totality of the problem. Thus, the following theorem is considered [63]:

Theorem 1: Assume *T* is an acyclic TBox (terminology) and *T′* is its expansion. Then:

1. *T* and *T′* have the same name and symbol.
2. *T* and *T′* are equivalent.
3. Both *T* and *T′* are definitorial.

Proof: T_1 is a terminology. Assume $A \equiv C$ and $B \equiv D$ are T_1 definitions that *B* occurs in *C*. By contrast, assume C' is a concept that is resulted from the substitution of each *B* repetition in *C* with *D* and assume that T_2 uses a terminology that is resulted from the substitution of $A \equiv C$ with $A \equiv C'$ in T_1.

Thus, both terminologies have the same name and symbol. Both terminologies have the same model because T_2 resulted from T_1 through a substitution.

Then, satisfiability and subsumption can be solved at the same time by considering the same TBox and ABox for a domain. The realization problem is formally defined before solving any of these problems. All the following formal definitions are extracted from [63].

Definition 1 (realization):
Given an ABox of *A*, Concept *C*, individual *a* and a set of concepts, find C as the *most specific concepts* from the set such that $A \vDash C(a)$.
An individual is called *a* and a collection of concepts are given. Find *c* (most specific concept) from the collection of concepts, such that $A \vDash C(a)$.
The definition of the most specific concept is as follows:

Definition 2 (most specific concept):
Assume *A* is an ABox in DL and *a* is an individual in *A*. *C* concept is called the most specific concept for *a*. Based on *A*, MSC(*A*,*a*) can be written for each *b* concept in DL. $A \nvDash D(a)$ implies that $C \subseteq D$.
Clearly, when MSC (*A*,*a*) is specified to determine whether *a* is an example of *D* concept, it is enough to check whether MSC(*A*,*a*) follows *D* concept.

As mentioned previously, the realization problem involves finding an individual that has the most relevance to a concept. Then, the difficulty regarding the reasoning of this problem involves finding the most relevance to a concept, which needs a full investigation of the TBox and ABox structures.

This study aims to solve the realization problem by solving subsumption and satisfiability, which are simple problems. First, the formal definitions are discussed. Then, the main method is presented.

Definition 3 (satisfiability):
Based on *T*, *C* concept is satisfiable if the I model exists in *T*, such that C^I is not empty.

Definition 4 (subsumption):
Let *T* be a TBox (terminology), a concept *C* is subsumed by a concept *D* with respect to *T* if $C^I \subseteq D^I$ for every model I of *T*. In this case, we write $C \sqsubseteq_T D$ or$T \vDash C \sqsubseteq D$.
By contrast, in specific conditions, the satisfiability problem may turn into the subsumption problem. Thus, solving the subsumption-like subcategories to meet the realization satisfaction is enough to solve the realization problem. Based on [63], satisfiability can be decreased into subsumption. Consider the following theorems:

Theorem 2: For *D* and *C* concepts:
C is unsatisfiable **if and only if** *C* is subsumed by empty concept ⊥.
Proof: Refer to [63].

Theorem 3: For *D* and *C* concepts:
C and *D* are equivalent if and only if *C* is subsumed by *D* and *D* is subsumed by *C*.
Proof: Refer to [63].
Based on the two previously presented problems, the realization problem can be solved solely through subsumption. Thus, for solving realization:

1. Assume *A* is the collection of concepts in realization and *a* is the individual.
2. The satisfiability problem for *A* collection is solved based on *a*, and the collection of concepts is assumed to satisfy *a* in *R* collection.
3. The subsumption is solved for all possible pairs of *R* collection, and MSC(*R*,*a*) is achieved.
4. According to the fact that satisfiability is reducible to subsumption, at the second level, subsumption is used to solve the satisfiability problem.

The main challenge is the efficiency of this method for collections with a large number of concepts. In the following part, an algorithm for reducing and optimizing the primary concept collection is presented. In other words, a solution to the ontology partitioning problem is derived. After partitioning the ontology into different sub-ontologies, the procedure for solving the realization problem for one of the sub-ontologies is performed.
In ontology partitioning, ontology O is partitioned into a collection of modules, that are not necessarily disjoint, such that the union of all modules is equal to O. The partition function is then formally defined as follows:

Definition 5 (ontology partitioning function):

$$Partition(O) \rightarrow M = \{\{M_1, M_2, \dots, M_n\} | \{M_1 \cup M_2 \cup \dots \cup M_n\} = O\}$$

According to the fact that correlation among concepts is achieved through the ontology structure, an acyclic graph is used for ontology partitioning.

In this case, graph O = (*C*,*D*), where *C* is the collection of concept and *D* is the dependency collection of concepts. Then, this procedure ontology is changed into a multipartite graph, such that the result of the query might exist in one or some parts of the graph. If the query results exist in one part of the graph, then only that part of the graph is investigated because of its independence from different parts. Other ontology parts are not checked.

The proof of the presented idea is as follows:

Let suppose that, after doing the graph partitioning, the related part of ABox *A* (with Tbox *T*) to query *q* is *p*. Then, we solve the satisfiability problem in p:

$$\exists\, I \in p, C^I \; is\; not\; \perp \qquad (1)$$

So, we find all *C* in p that are satisfiable and put them all in *R*:

$$R = \{R_1, R_2, \dots, R_n\}, R_i \; is\; satisfiable\; concept\; in\; p \qquad (2)$$

Then, for all pairs of satisfiable concepts in *R*, we solve subsumption problem as follows:

$$\forall (i1, i2) \in R, \forall\, I \in T \qquad C^I \sqsubseteq D^I \qquad (3)$$

If *i1* is subsumptive to *i2*, then *i1* is removed from *R*. We apply relation (3) for every pair of *i1* and *i2* since there is no candidate *i1* and *i2* and we name new *R* as R'. Then we apply all concepts in R' to individual a such that:

$$\forall C \in R' \quad if\; A \not\models C(a)\; then\; remove\; C\; from\; R' \qquad (4)$$

So R' has a candidate $msc(A, a)$ such that the real most specific concept *A* and a is in R'.

On the other hand, according to theorem 1,2,3 and [63] we can solve satisfiable problem (relation (1)) using only the subsumption problem. So we find candidate solutions for realization problem in reasonable time and we use only the subsumption problem.

The general problem solving procedure is the same as it is shown in Figure 1. The pseudo code of the algorithm is shown in Figure 2.

In line 3 from Fig 2, we used ontology partitioning algorithm based on [64] approach which describes that using the pseudo code format in Fig 1. After partitioning ontology into parts and selecting a relevant part(s) in the ontology partitioning phase, the important point is to combine and merge query answers from relevant parts to produce the final answer.
In other words, we must prove that the final answer can be produced from instance checking through independent Abox. Before the definition of respective theorem, we should declare some relevant terms [65].

Definition 6: Abox graph
One Abox graph for Abox A, is AG (A) that contains vertex set V, edge set U and the following function:

$$\mathrm{F: E} \rightarrow \{ (\mathrm{a}, \mathrm{b}) |\, \mathrm{a}, \mathrm{b} \in \mathrm{A}\} \qquad (5)$$

Each vertex is related to one individual and AG (A) is directed as a multipartite graph.

Definition 7: (Abox dependency)
Two connected Abox A_1 and A_2 are given such that$\mathrm{A} = \mathrm{A}_1 \cup \mathrm{A}_2$. If $\mathrm{A}_1, \mathrm{A}_2$ are dependent, Abox graph A is connected, and If $\mathrm{A}_1, \mathrm{A}_2$ are independent, Abox graph A is disconnected.

So, we can declare the following theorem.

Theorem 4: Independent Abox and Instance checking
Two connected Abox A_1 and A_2 are given such that$\mathrm{A} = \mathrm{A}_1 \cup \mathrm{A}_2$. If $\mathrm{A}_1, \mathrm{A}_2$ are independent then for each query realization, we have Φ and Tbox T:

$< \mathrm{T}, \mathrm{A} > \models\ \Phi$ if and only if $< \mathrm{T}, \mathrm{A}_1 > \models \Phi$ or $< \mathrm{T}, \mathrm{A}_2 > \models \Phi$.

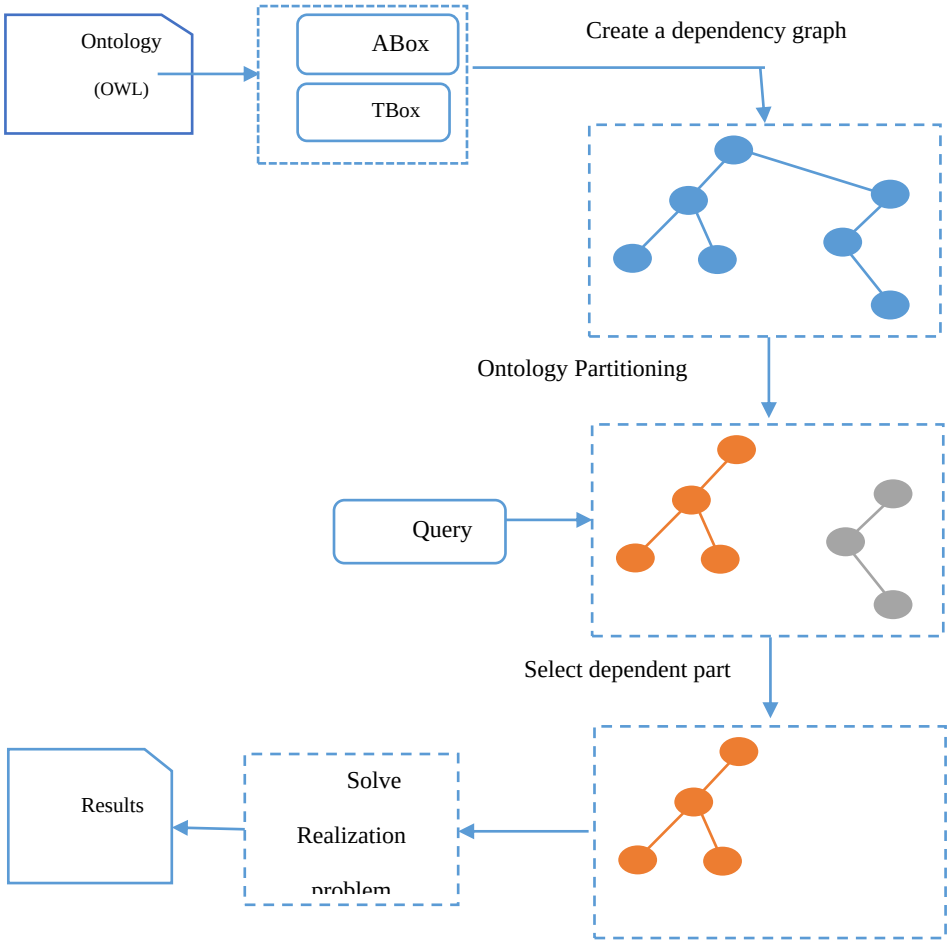

Fig.1: Overall proposed process

Algorithm 1 Realization Solving

```
1: procedure ONTOLOGY—PARTITIONING(ABOX A, QUERY q, INDIVIDUAL a)
2:     Make ontology partitioning using Overlapped − Ontology − Partitioning(A)
3:     Select dependent part p according to q from partitions
4:     Solve Realization according to p
5:     return Solve − Realization(p,q,a)
6: end procedure
7: procedure SOLVE—REALIZATION(ABOX p, QUERY q, INDIVIDUAL a)
8:     Solve Satisfiability(p,q,a) and store results in R
9:     while no subsumption problem exists do
10:        for each pair of items (i1,i2) in R do
11:            Solve subsumption(i1,i2) problem and store results in S
12:        end for
13:    end while
14:    return S as a result
15: end procedure
```

Fig.2: Proposed algorithm for the realization problem

Algorithm 2 Ontology Partitioning

```
procedure OVERLAPPED—ONTOLOGY—PARTITIONING(ONTOLOGY O)
    Building the weighted dependency graph WG from an ontology O
    Finding the common concepts in WG
    Partitioning the weighted graph WG
    Using Rank Removal Algorithm for cluster components
    Extracting partitioned ontologies as PO set
    return PO
end procedure
```

Fig.3: Overlapped Ontology Partitioning

Proof:

(→)

Suppose that A_1 and A_2 are independent and $\Delta^I_{A_1}, \Delta^I_{A_2}$ are domains of A_1 and A_2, then:

$$\Delta^I_{A_1} \cap \Delta^I_{A_2} = \emptyset \quad (6)$$

For each concept C, $C^I_{A_1} \cap C^I_{A_2} = \emptyset$ which $C^I_{A_i}$ is extended C in $\Delta^I_{A_i}$.

On the other hand, suppose that $< T, A_1 > \not\models Q$ and $< T, A_2 > \not\models Q$ which means:

$$\exists I_1; I_1 \models A_1, I_1 \models T, I_1 \models \neg Q \quad (7)$$

$$\exists I_2; I_2 \models A_2, I_2 \models T, I_2 \models \neg Q \quad (8)$$

Which I_1 and I_2 are explanations of A_1 and A_2.

Since $A = A_1 \cup A_2$ and $\Delta^I_1 \cup \Delta^I_2 = \emptyset$, we can create the explanation from A like I which $I = I_1 \cup I_2$. In other words $I =< \Delta^I, O^I >$ that declare as follows:

(i) $\Delta^I = \Delta^I_1 \cup \Delta^I_2$

(ii) For any constant a, $a^I = \begin{Bmatrix} a^{I_1} \text{ if a occurs in } A_1 \\ a^{I_2} \text{if a occurs in } A_2 \end{Bmatrix}$

(iii) for any concept C,$C^I = C^{I_1} \cup C^{I_2}$

(iv) For any role R, $R^I = R^{I_1} \cup R^{I_2}$

So, we can conclude from $I_1 \models \neg Q$, $I_2 \models \neg Q$ and (iii):

$$I \models \neg Q \quad (9)$$

Which means:

$$(\neg Q)^I = (\neg Q)^{I_1} \cup (\neg Q)^{I_2} \quad (10)$$

That I is an explanation of A. On the other hand we can conclude from (ii), (iii) and (iv) that:

$$(A)^I = (A_1)^{I_1} \cup (A_1)^{I_2} \quad (11)$$

Because A_1 and A_2 are consistent, we just prove no intersection between them.

For Concept C from DL, we have:

$$C^I \subseteq C^I \quad (12)$$

On the other hand, we have:

$$(\neg C)^I = (\Delta^I \backslash \Delta^I) \subseteq \Delta^I \quad (13)$$

Then, for C, we have:

$$C^{I_1} \subseteq \Delta^I_1 \text{ and } (\neg C)^{I_2} \subseteq \Delta^I_2 \quad (14)$$

Because of $\Delta^I_1 \cap \Delta^I_2 = \emptyset$, then:

$$C^{I_1} \cap (\neg C)^{I_2} = \emptyset \quad (15)$$

This means they have no intersection. Since, we have the explanation I from A:

Since, we have the explanation I from A:

$$(A)^I \neq Q \text{ and } (\neg Q)^T \neq Q \quad (16)$$

Then:

$$I \models A \cap I \models \neg Q \quad (17)$$

Which is an $A \not\models Q$ definition.

Therefore, $A \models Q$ if $A_1 \models Q$ or $A_2 \models Q$ that result is $<T,A> \models Q$ if $<T,A_1> \models Q$ or $<T,A_2> \models Q$.

(←)

We suppose that $<T,A_1> \models Q$ or $<T,A_2> \models Q$. In both cases we have:

$$<T,A_1 \cup A_2> \models Q \quad (18)$$

Which $<T,A> \models Q$. ■

5.0 ILLUSTRATION THE EXAMPLE

In this section, we describe the proposed algorithm and give an example of identifying the concept that is closest to the query, i.e., the realization concept.

Step 1:

In this step, the ontology is partitioned, and the presented ontology is divided into separate parts. The solution to the realization problem is not obtained from the whole ontology but from the partitions associated with the query.

In this example, we use the automotive production ontology. The following partitions are produced created after partitioning the ontology.

Step 2:

Suppose the query is as follows:

"When was the car with a 3500 cc engine produced?"

First, the term 3500 cc is searched in the ontology. Then the concept is identified in the sub-ontology of "car manufacturer B." So, the remaining steps of this algorithm are performed only in this sub-ontology.

Step 3:

The solution to the realization problem in the sub-ontology of car production B is discussed.

Let us assume that the structure is as shown in the following figure.

Step 4:

The satisfiability problem is solved for car production B. Thus, the set of concepts that concludes car production B or the concepts that satisfy A must be identified.

Therefore, R is obtained as follows:

R = {Iran khodro, products of 2004, '206' and petrol and chassis and motor}

Car production (A) **Car production (b)** **Car production (C)**

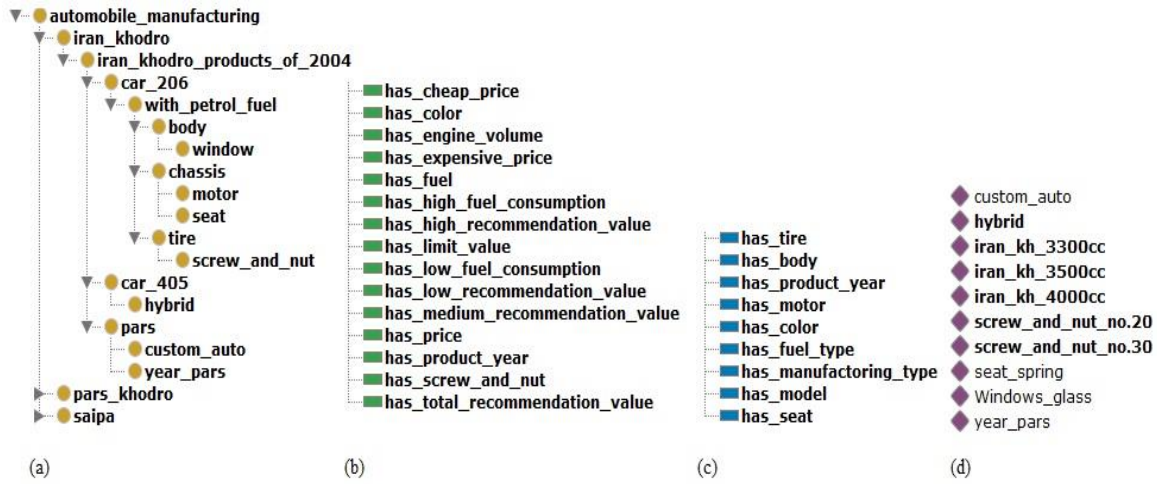

Fig.4: An excerpt of the Car Manufacturing model indicating (a) Classes, (b) Data Type Properties, (c) Object Properties and (d) Individual

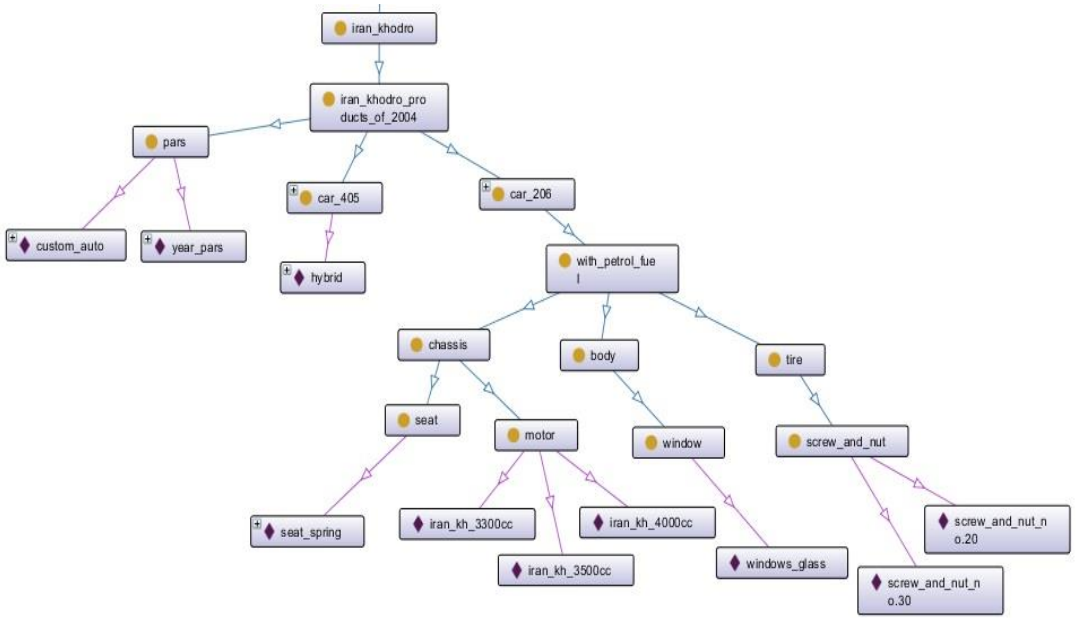

Fig.5: Car Manufacturing Ontology

Step 5:

The subsumption problem is implemented as all possible pairs in the R set. See Fig. 6.
Therefore, the most specific concept has the following elements:
Msc (Iran khodroo, 3500 cc) = (206, products of 2004).

Step 6:

As proven by Theorem 4, the final answer to the query is the sum of this response.
Answer: {206, products of 2004)

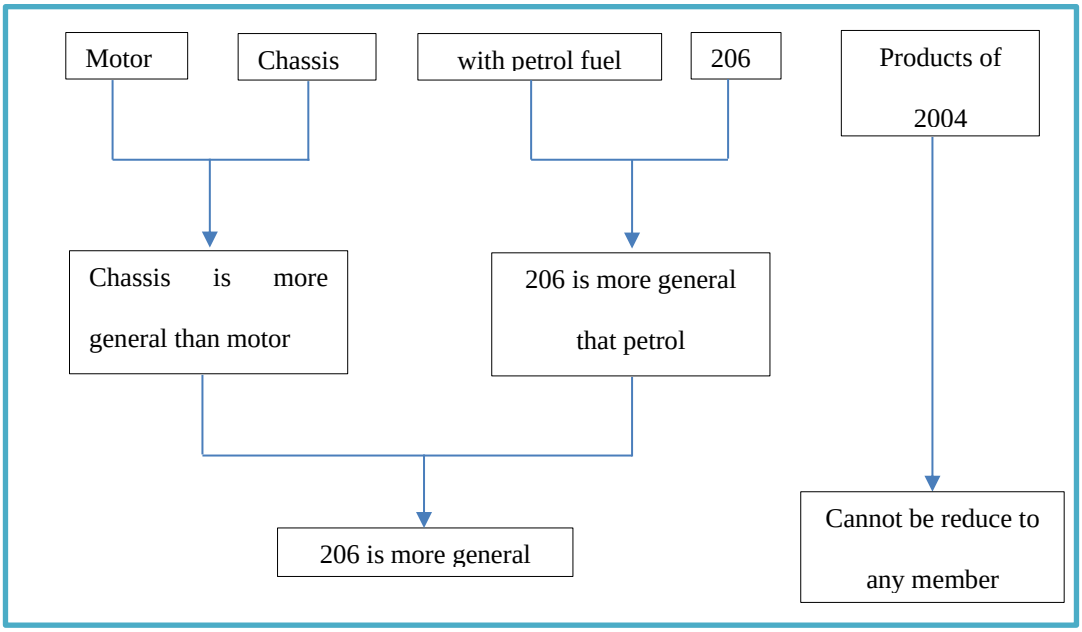

Fig.6: Subsumption problem

6.0 RESULT ANALYSIS

The method proposed in [66], which has a standard structure for the evaluation of ontology reasoning is used to assess the proposed method. The ORE framework presented in [66] is the framework for the OWL reasoner evaluation. The framework is tuned for UNIX-based operating systems. The ORE framework can be applied to any given reasoner and set of ontologies. Each reasoner my implement in the specific folder structure to evaluate with this framework. It supports two components for test and verifies the reasoners. The reasoner presented in this article are customized with the mentioned specific structure of the ORE framework and the results of the ORE test output are reported below.

The data set is used in the framework which was produced by the ontology Reasoner Evaluation Workshop in 2014 [67]. The criteria for assessment of the introduced reasoners in this framework are as follows:

1. Realization
2. Classification
3. Consistency

The proposed method is compared with the HermiT [30] and FaCT++[31] reasoners based on the aforementioned parameters . The JFaCT reasoner is the JAVA implementation of the FaCT++ reasoner, and their differences are presented in Table 4[67]**.**

Table 4: Comparison between JFaCT and FaCT++

Characteristics	**JFaCT**	**FaCT++**
OWL2 profile supported	EL, RL, QL, DL	
Interfaces	OWLAPI	OWLAPI, LISP
Algorithms	Tableaux	Tableaux
Optimizations	Same as FaCT++	N/A
Advantages	Pure Java; extended DT	good general performance
Disadvantages	Work in progress	OWLAPI interface is complicated
Application focus	General purposes	

The results of the comparison between the proposed method and the above reasoners based on three parameters are shown in Tables 5 to 7.

The scores in the tables indicate the accuracy of the answers given by the reasoner. In other words, a score of 253 out of 264 means that 253 out of 264 questions were answered correctly. The error rate shows how many wrong answers were given by the reasoner.

The tables clearly show that the proposed method outperforms the other ones in terms of the three parameters.

Table 5: Comparison of different methods based on the Realization parameter

Rank	**Reasoner**	**Score**	**Error**	**Time(s)**
1	Proposed Reasoner	253/264	11	545.68 s
2	FaCT++	172/264	92	1111.3 s
3	HermiT	163/264	101	2934.9 s
4	HermiT-OA4	162/264	102	3022.5 s

Table 6: Comparison of different methods based on the classification parameter

Rank	**Reasoner**	**Score**	**Error**	**Time(s)**
1	Proposed Reasoner	292/306	14	1318.18 s
2	HermiT-OA4	237/306	69	5808.2 s
3	HermiT	236/306	70	5416.4 s
4	FaCT++	200/306	106	1361.3 s

Table 7: Comparison of different methods based on the consistency parameter

Rank	Reasoner	Score	Error	Time(s)
1	Proposed Reasoner	300/306	6	1501.6 s
2	HermiT	294/306	12	1449.6 s
3	HermiT-OA4	293/306	13	1549.4 s
4	FaCT++	276/306	30	1341.2 s

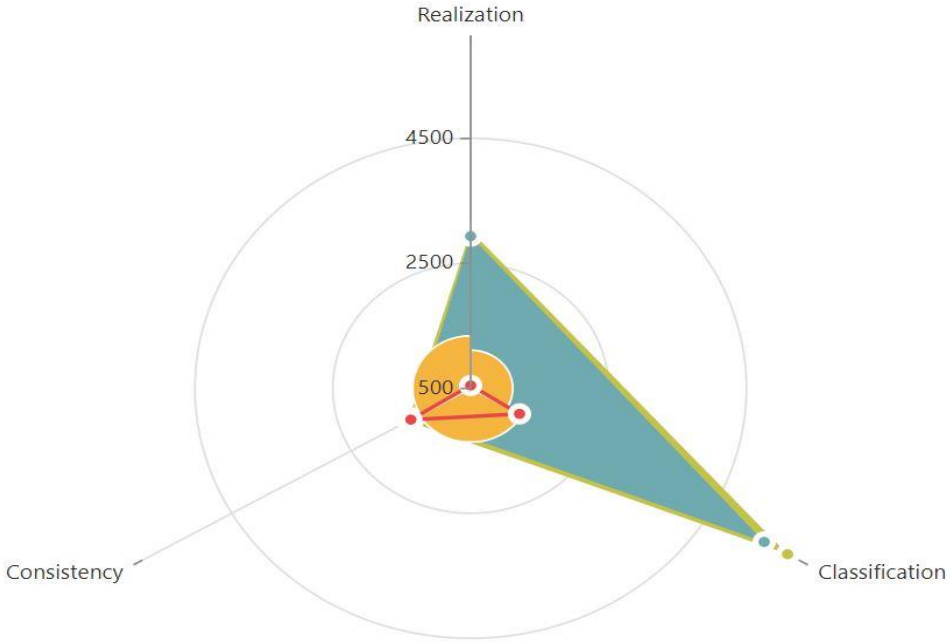

Fig.7: Comparison of elapsed average time for reasoners in seconds

7.0 CONCLUSION

This study proposed an efficient method to solve the realization problem based on the satisfiability and subsumption problems. The computational complexity is the main problem in solving the realization problem. The proposed method compensated this drawback to a certain extent. The proposed method is implemented in the form of a standard reasoner and it is compared with other reasoners in a standard framework. The results revealed an optimized performance compared to previous studies. For conducting future studies, the main challenge is to overcome the time complexity of the realization problem that needs different methods, such as random probabilistic, approximate, genetic, and exploratory algorithms.

REFERENCES

[1] C. Elsenbroich, O. Kutz, U. Sattler, A Case for Abductive Reasoning over Ontologies, in: OWLED, (2006).

[2] C. Bettini, O. Brdiczka, K. Henricksen, J. Indulska, D. Nicklas, A. Ranganathan, D. Riboni, A survey of context modelling and reasoning techniques, Pervasive and Mobile Computing, 6 (2010) 161-180.

[3] I. Horrocks, L. Li, D. Turi, S. Bechhofer, The instance store: DL reasoning with large numbers of individuals, in: Proc. of the 2004 Description Logic Workshop (DL 2004), (2004), pp. 31-40.

[4] B. Glimm, I. Horrocks, B. Motik, G. Stoilos, HermiT: reasoning with large ontologies, Computing Laboratory, Oxford University, (2009).

[5] D. Tsarkov, I. Horrocks, FaCT++ description logic reasoner: System description, in: International Joint Conference on Automated Reasoning, (Springer, 2006), pp. 292-297.

[6] Y. Welsch, A. Poetzsch-Heffter, A fully abstract trace-based semantics for reasoning about backward compatibility of class libraries, Science of Computer Programming, 92 (2014) 129-161.

[7] C.C. Din, E.B. Johnsen, O. Owe, I.C. Yu, A modular reasoning system using uninterpreted predicates for code reuse, Journal of Logical and Algebraic Methods in Programming, 95 (2018) 82-102.

[8] Bahadorani, B., & Zaeri, A. (2020). A method for using temporal reasoners to answer the history of science questions. International Journal of Information Technology, 12(1), 181-188.

[9] S. Singh, R. Karwayun, A comparative study of inference engines, in: Information Technology: New Generations (ITNG), 2010 Seventh International Conference on, (IEEE, 2010), pp. 53-57.

[10] X. Su, L. Ilebrekke, A comparative study of ontology languages and tools, in: International Conference on Advanced Information Systems Engineering, (Springer, 2002), pp. 761-765.

[11] Sarker, M. K., & Hitzler, P. (2019, July). Efficient concept induction for description logics. In Proceedings of the AAAI Conference on Artificial Intelligence (Vol. 33, No. 01, pp. 3036-3043).

[12] Carral, D., Dragoste, I., & Krötzsch, M. (2020). Reasoner= logical calculus+ rule engine. KI-Künstliche Intelligenz, 34(4), 453-463.

[13] Mehri, R., Haarslev, V., & Chinaei, H. (2018). Optimizing heuristics for tableau-based owl reasoners. arXiv preprint arXiv:1810.06617.

[14] S.D. Ravana, P. Samimi, P. Rajagopal, QUALITY OF CROWDSOURCED RELEVANCE JUDGMENTS IN ASSOCIATION WITH LOGICAL REASONING ABILITY, Malaysian Journal of Computer Science, (2018) 73-86.

[15] C. d'Amato, F. Esposito, N. Fanizzi, B. Fazzinga, G. Gottlob, T. Lukasiewicz, Inductive reasoning and semantic web search, in: Proceedings of the 2010 ACM Symposium on Applied Computing, (ACM, 2010), pp. 1446-1447.

[16] P. Kapłański, A. Seganti, K. Cieśliński, A. Chrabrowa, I. Ługowska, Automated reasoning based user interface, Expert Systems with Applications, 71 (2017) 125-137.

[17] F. Lashkari, F. Ensan, E. Bagheri, A.A. Ghorbani, Efficient indexing for semantic search, Expert Systems with Applications, 73 (2017) 92-114.

[18] R.A. Kadir, R.A. Yauri, A. Azman, SEMANTIC AMBIGUOUS QUERY FORMULATION USING STATISTICAL LINGUISTICS TECHNIQUE, Malaysian Journal of Computer Science, (2018) 48-56.

[19] Quan, Z., & Haarslev, V. (2019). A framework for parallelizing OWL classification in description logic reasoners. arXiv preprint arXiv:1906.07749.

[20] O. Pivert, O. Slama, G. Smits, V. Thion, A fuzzy extension of SPARQL for querying gradual RDF data, in: Research Challenges in Information Science (RCIS), 2016 IEEE Tenth International Conference on, (IEEE, 2016), pp. 1-2.

[21] S. Vigneshwari, SEFOS: Semantic enriched fuzzy based ontological integration of web data tables, in: Circuit, Power and Computing Technologies (ICCPCT), 2015 International Conference on, (IEEE, 2015), pp. 1-4.

[22] M. Joseph, G. Kuper, T. Mossakowski, L. Serafini, Query answering over contextualized RDF/OWL knowledge with forall-existential bridge rules: decidable finite extension classes, Semantic Web, 7 (2016) 25-61.

[23] P. Ristoski, E.L. Mencía, H. Paulheim, A hybrid multi-strategy recommender system using linked open data, in: Semantic Web Evaluation Challenge, (Springer, 2014), pp. 150-156.

[24] I. Fister, X.-S. Yang, K. Ljubič, D. Fister, J. Brest, I. Fister, Towards the novel reasoning among particles in PSO by the use of RDF and SPARQL, The Scientific World Journal, 2014 (2014).

[25] D.D. Gessler, E. Sirin, SSWAP: Enabling Transaction-Time Reasoning for Semantic Workflows, Computer, 48 (2015) 60-68.

[26] A. Lukasová, M. Zácek, English grammatical rules representation by a meta-language based on RDF model and predicate clausal form, International Information Institute (Tokyo). Information, 19 (2016) 4009.

[27] K. Kaneiwa, R. Mizoguchi, P.H. Nguyen, A Logical and Ontological Framework for Compositional Concepts of Objects and Properties, New Generation Computing, 33 (2015) 149-172.

[28] T.H.H. Nguyen, N. Le Thanh, Coloured Petri Nets-based Approach for Manipulating RDF Data, Journal of Automation and Control Engineering (JOACE), 3 (2014) 2301-3702.

[29] T.H.H. Nguyen, N. Le Thanh, An ontology-enabled approach for modelling business processes, in: 10th IEEE International Conference Beyond Databases, Architectures, and Structures (BDAS 2014), (Springer, 2014).

[30] A. Agostini, C. Bettini, D. Riboni, Online ontological reasoning for context-aware internet services, (2006).

[31] I. Horrocks, P.F. Patel-Schneider, H. Boley, S. Tabet, B. Grosof, M. Dean, SWRL: A semantic web rule language combining OWL and RuleML, W3C Member submission, 21 (2004) 79.

[32] F. Riguzzi, E. Bellodi, E. Lamma, R. Zese, BUNDLE: A reasoner for probabilistic ontologies, in: International Conference on Web Reasoning and Rule Systems, (Springer, 2013), pp. 183-197.

[33] F. Baader, C. Lutz, B. Suntisrivaraporn, CEL—a polynomial-time reasoner for life science ontologies, in: International Joint Conference on Automated Reasoning, (Springer, 2006), pp. 287-291.

[34] T. Eiter, M. Ortiz, M. Simkus, T.-K. Tran, G. Xiao, Query Rewriting for Horn-SHIQ Plus Rules, in: AAAI, (2012).

[35] M. del Mar Roldan-Garcia, J.F. Aldana-Montes, DBOWL: Towards a Scalable and Persistent OWL reasoner, in: Internet and Web Applications and Services, 2008. ICIW'08. Third International Conference on, (IEEE, 2008), pp. 174-179.

[36] F. Bobillo, M. Delgado, J. Gómez-Romero, Reasoning in fuzzy OWL 2 with DeLorean, in: Uncertainty Reasoning for the Semantic Web II, (Springer, 2013), pp. 119-138.

[37] J. Baker, The DRAGON system--An overview, IEEE Transactions on Acoustics, Speech, and Signal Processing, 23 (1975) 24-29.

[38] A. Corbel, J.-J. Girardot, P. Jaillon, Drew: A dialogical reasoning web tool, in: Int. conf. on ict's in education, (2002).

[39] J. Mendez, jcel: A Modular Rule-based Reasoner, in: ORE, (2012).

[40] A. Steigmiller, T. Liebig, B. Glimm, Konclude: system description, Web Semantics: Science, Services and Agents on the World Wide Web, 27 (2014) 78-85.

[41] D. Tsatsou, S. Dasiopoulou, I. Kompatsiaris, V. Mezaris, LiFR: A lightweight fuzzy DL reasoner, in: European Semantic Web Conference, (Springer, 2014), pp. 263-267.

[42] A.A. Romero, B.C. Grau, I. Horrocks, MORe: Modular combination of OWL reasoners for ontology classification, in: International Semantic Web Conference, (Springer, 2012), pp. 1-16.

[43] I. Horrocks, U. Sattler, Ontology reasoning in the SHOQ (D) description logic, in: IJCAI, (2001), pp. 199-204.

[44] Y. Kazakov, M. Krötzsch, F. Simancik, ELK Reasoner: Architecture and Evaluation, in: ORE, (2012).

[45] F. Bobillo, U. Straccia, fuzzyDL: An expressive fuzzy description logic reasoner, in: FUZZ-IEEE, (2008), pp. 923-930.

[46] R. Mutharaju, P. Hitzler, P. Mateti, DistEL: A distributed EL+ ontology classifier, in: Proceedings of the 9th International Conference on Scalable Semantic Web Knowledge Base Systems-Volume 1046, (CEUR-WS. org, 2013), pp. 17-32.

[47] D. Tsarkov, I. Palmisano, Chainsaw: a Metareasoner for Large Ontologies, in: ORE, (2012).

[48] C.J. Matheus, K. Baclawski, M.M. Kokar, Basevisor: A triples-based inference engine outfitted to process ruleml and r-entailment rules, in: 2006 Second International Conference on Rules and Rule Markup Languages for the Semantic Web (RuleML'06), (IEEE, 2006), pp. 67-74.

[49] B. Parsia, E. Sirin, Pellet: An owl dl reasoner, in: Third International Semantic Web Conference-Poster, (2004).

[50] K. Wu, V. Haarslev, Parallel owl reasoning: Merge classification, in: Joint International Semantic Technology Conference, (Springer, 2013), pp. 211-227.

[51] B. Sertkaya, The ELepHant Reasoner System Description, in: ORE, (2013), pp. 87-93.

[52] E. Friedman-Hill, Jess, the rule engine for the java platform, in, (2008).

[53] F. Bobillo, U. Straccia, Finite fuzzy description logics and crisp representations, in: Uncertainty Reasoning for the Semantic Web II, (Springer, 2010), pp. 99-118.

[54] J. Dolby, A. Fokoue, A. Kalyanpur, E. Schonberg, K. Srinivas, Scalable highly expressive reasoner (SHER), Web Semantics: Science, Services and Agents on the World Wide Web, 7 (2009) 357-361.

[55] I. Horrocks, FaCT and iFaCT, Description logics, 22 (1999).

[56] D. Tsarkov, I. Horrocks, FaCT++ description logic reasoner: System description, in: Automated reasoning, (Springer, 2006), pp. 292-297.

[57] V. Haarslev, K. Hidde, R. Möller, M. Wessel, The RacerPro knowledge representation and reasoning system, Semantic Web, 3 (2012) 267-277.

[58] A. Jena, Reasoners and rule engines: Jena inference support, The Apache Software Foundation, (2013).

[59] Y. Zou, T. Finin, H. Chen, F-owl: An inference engine for semantic web, in: Formal Approaches to Agent-Based Systems, (Springer, 2005), pp. 238-248.

[60] J. Zhao, H. Boley, Uncertainty Treatment in the Rule Interchange Format: From Encoding to Extension, in: URSW, (2008).

[61] A. Kiryakov, OWLIM: balancing between scalable repository and light-weight reasoner, Proc. of WWW2006, Edinburgh, Scotland, (2006).

[62] C.J. Matheus, K. Baclawski, M.M. Kokar, Basevisor: A triples-based inference engine outfitted to process ruleml and r-entailment rules, in: Rules and Rule Markup Languages for the Semantic Web, Second International Conference on, (IEEE, 2006), pp. 67-74.

[63] F. Baader, The description logic handbook: Theory, implementation and applications, (Cambridge university press, 2003).

[64] K. Etminani, A.R. Delui, M. Naghibzadeh, Overlapped ontology partitioning based on semantic similarity measures, in: 2010 5th International Symposium on Telecommunications, (2010), pp. 1013-1018.

[65] P. Pothipruk, G. Governatori, A formal ontology reasoning with individual optimization: a realization of the semantic web, in: International Conference on Web Information Systems Engineering, (Springer, 2005), pp. 119-132.

[66] R.S. Gonçalves, S. Bail, E. Jiménez-Ruiz, N. Matentzoglu, B. Parsia, B. Glimm, Y. Kazakov, OWL Reasoner Evaluation (ORE) Workshop 2013 Results: Short Report, in: ORE, (2013), pp. 1-18.

[67] N.a.P. Matentzoglu, Bijan, ORE 2014 Reasoner Competition Dataset, (2014).