IEICE TRANS. INF. & SYST., VOL.E101–D, NO.1 JANUARY 2018

PAPER Special Section on Semantic Web and Linked Data

A Comparative Study of Rule-Based Inference Engines for the Semantic Web

Thanyalak RATTANASAWAD[†], **Marut BURANARACH**[†a)], **Kanda Runapongsa SAIKAEW**[††], ***Nonmembers***, ***and*** **Thepchai SUPNITHI**[†], ***Member***

SUMMARY With the Semantic Web data standards defined, more applications demand inference engines in providing support for intelligent processing of the Semantic Web data. Rule-based inference engines or rule-based reasoners are used in many domains, such as in clinical support, and e-commerce recommender system development. This article reviews and compares key features of three freely-available rule-based reasoners: Jena inference engine, Euler YAP Engine, and BaseVISor. A performance evaluation study was conducted to assess the scalability and efficiency of these systems using data and rule sets adapted from the Berlin SPARQL Benchmark. We describe our methodology in assessing rule-based reasoners based on the benchmark. The study result shows the efficiency of the systems in performing reasoning tasks over different data sizes and rules involving various rule properties. The review and comparison results can provide a basis for users in choosing appropriate rule-based inference engines to match their application requirements.

key words: *rule language, rule-based reasoner, benchmark*

1. Introduction

With the Semantic Web data standards defined, metadata and ontologies, i.e., Resource Description Framework (RDF), and Web Ontology Language (OWL) data, are increasingly published on the Web. In addition, the Semantic Web stack [1] emphasizes the need for rule language for the Web. The rule language can enhance the ontology language by allowing one to describe relations that cannot be described using Description Logic (DL) used in OWL. For example, rule language can allow for inference of property value assignment, e.g. *hasParent(?x,?y)^hasBrother(?y,?z)* → *hasUncle(?x,?z)*, which is impossible in DL. Processing of rules typically demands inference engines in providing support for an intelligent application. Rule-based inference engines or rule-based reasoners are used in several domains, such as clinical support, and e-commerce recommender system development. Functions of rule-based inference engines for the Semantic Web typically include high-performance reasoning algorithms, compatibility with the Semantic Web standards, providing interchangeable syntax and supporting expressive rule languages with built-in functions.

This article conducts a comparative study of three rule-based reasoners designed for the Semantic Web: Jena Inference Engine, Euler YAP Engine (EYE), and BaseVISor. The main criteria for selecting the systems for the study were based on the following characteristics: available under free software or freeware license, having an active user community, or updated publications or software. While some researchers studied and compared semantic reasoners [2], [3], our study focuses on the features of rule-based reasoners designed for the Semantic Web data.

In addition to system features, reasoning performance is one of the most important factors in evaluating a rule-based inference engine. Different factors of rules can affect the rule-based system performance including join complexity, production operation, negation, built-in functions, and rule dependency. Our study investigated the performance of the reviewed rule-based reasoners in terms of response time in performing rule-based reasoning task. We describe an evaluation methodology to assess the scalability and efficiency of rule-based reasoners using the data and rule sets adapted from the Berlin SPARQL Benchmark (BSBM) [4]. The comparison of features and benchmark result can guide users in selecting rule-based inference engines.

2. Review and Comparison of Rule-Based Inference Engines for the Semantic Web

Inference engines (or reasoners) are application software for computing or deriving new facts from existing knowledge bases. Although there are various types of inference engines, our study focuses on only rule-based inference engines. A rule-based inference engine applies rules with the data to reason and derive some new facts. When the data matches with rule conditions, the inference engine can modify the knowledge base, e.g., fact assertion or retraction, or execute functions, e.g., displaying the derived facts. Our review focuses on rule-based reasoners aimed for the Semantic Web data processing.

2.1 Comparison Criteria

Some comparison criteria for assessing rule-based reasoners for the Semantic Web are discussed as follows.

Reasoning Strategies and Algorithms. Reasoning strategies are methods used by an inference engine to perform reasoning tasks. There are two main strategies of reasoning, for-

Manuscript received February 13, 2017.
Manuscript revised June 2, 2017.
Manuscript publicized September 15, 2017.

[†]The authors are with the National Electronics and Computer Technology Center (NECTEC), Thailand.
[††]The author is with Faculty of Engineering, Khon Kaen University, Thailand.
a) E-mail: marut.bur@nectec.or.th

DOI: 10.1587/transinf.2017SWP0004

ward chaining, and backward chaining [5]. Forward chaining starts with existing facts and applying rules to derive all possible facts, while backward chaining starts with the desired conclusion and performs backward to find supporting facts. Optimized algorithms and techniques may be used to improve the performance of the reasoning process.

Expressivity for Reasoning. Inference engines may support different levels or subsets of logics in reasoning. For the Semantic Web data, the common levels of reasoning are typically RDFS, various subsets of OWL, and user-defined rules. RDFS and OWL reasoning support is typically configured as a set of pre-defined inference rules which support some specified semantics. For example, RDFS reasoning support may be provided as a set of rules that covers the model-theoretic semantics of RDFS [6]. User-defined rules are normally defined as a set of custom inference rules that can be applied in addition to the pre-defined RDFS and OWL rules.

Built-in Functions and User-defined Functions. Built-in functions are functions provided for calculating values of variables in rule clauses. Some common built-in functions include mathematical, boolean, and string functions. Some inference engines may provide supports for users to additionally create custom functions.

Reasoning Features. In addition to performing inference and deriving new facts, an inference engine may provide some additional useful functions, such as proof explanation, and proof tracing. An inference engine may also provide a number of configuration options of reasoning, for example, caching, and output filtering.

Supported Rule Languages. Rules for the Semantic Web data can be expressed in various languages and formats [7]. Inference engines either support standard rule languages, e.g., Rule Interchange Format (RIF) [8], Notation3 [9], or their own rule languages, e.g., Jena rules [10].

2.2 Comparison of Rule-Based Inference Engines

This section reviews three rule-based reasoners aimed for the Semantic Web data processing: Jena Inference Engine, EYE, and BaseVISor. Although some systems may provide other functions beyond inference engines, i.e. semantic repository, programming environment, etc., our review of the systems only focuses on its inference engine sub-system. The systems selected for the review were based on the following characteristics. The software must be available under free software or freeware license. It must have some user community or have updated publications or software within the past three years. These requirements are generally defined to limit the scope of the studied systems to more available and active systems. It should be noted that the systems included in the study were selective and not intended to be comprehensive.

Jena Inference Engine. Jena Inference Engine [10] is one of the development tools of the Jena framework [11], a Java-based open-source application framework for developing Semantic Web applications adopted by the Apache Software Foundation. The framework provides a number of predefined reasoners including an RDFS reasoner, an OWL-lite reasoner, and a general-purpose rule reasoner supporting user-defined rules in the Jena own syntax. The rule reasoner supports three execution strategies: forward-chaining, tabled backward-chaining, and hybrid. The rule reasoner consists of two internal rule engines, a forward engine based on the RETE [12] algorithm and a tabled Datalog backward engine. A number of primitive built-in functions are provided and can be extended by users. Additional reasoning features include proof tracing, proof explanation, and preprocessing attachment.

Euler YAP Engine. EYE (Euler YAP Engine) [13], [14], is a backward-chaining inference engine based on underlying Prolog YAP engine [15], a high-performance Prolog compiler for demand-driven indexing. The inference engine uses backward-forward-backward chaining reasoning cycle and Euler path detection algorithm to avoid loops in an inference graph. EYE can be configured with many options of reasoning, e.g., not proving false model, output filtering, and providing useful information of reasoning, e.g., proof explanation, debugging and warning logs. The inference engine also supports using user-defined plugins. In addition to the main supported language, Notation3, EYE also supports the RIF-BLD [16] (Basic Logic Dialect) rule language.

BaseVISor. BaseVISor [17], [18], developed by VIStology, is a forward chaining inference engine based on RETE network optimized for the processing of RDF triples. The inference engine supports OWL2-RL and rule-based reasoning, and supports rules in its own format. The priority of a rule may be specified. The inference engine supports operations including fact assertion, retraction, and procedural control functions, e.g. halting and throwing exception. It also supports external procedural attachments. Some optimizations are provided, e.g. indexing, caching, join optimization. The engine supports a number of built-in functions, which can also be extended by users.

The features of the rule-based reasoners were compared based on the following software versions: Jena 2.10.1, EYE 2013-05, and BaseVisor 2.0. Table 1 presents a comparison of the features related to reasoning strategies and algorithms of the system, supported RDFS/OWL reasoning, rule languages [7], reasoning operations and features provided, i.e. built-in and user-defined function, proof explanation facility and software license.

There are some additional inference engines providing support for rule-based inference over the Semantic Web data which are actively being developed. Openllet[†] is an

[†]https://github.com/Galigator/openllet

Table 1 Comparison of the features of reviewed rule-based inference engines

	Jena	EYE	BaseVISor
Reasoning Strategies			
Forward Chaining	✓	✓	✓
Backward Chaining	✓	✓	-
Reasoning Algorithms	RETE	Euler Path Detection, YAP Prolog engine	RETE
RDFS/OWL Reasoning	RDFS, OWL Lite	selected predicates of RDFS and OWL	OWL2-RL
Rule Languages	Own format	Notation3, RIF-BLD	Own format
Production Operations			
Assertion	✓	✓	✓
Retraction	✓	-	✓
Built-in Functions	✓	✓	✓
User-defined Functions	✓	✓	✓
Proof Explanation	✓	✓	-
Software License	Apache License 2.0	MIT License	Free for academic/ research use

open-source continuation of Pellet [19], an OWL2 DL Reasoner, which supports reasoning with SWRL (Semantic Web Rule Language) rules. The SWRLAPI Drools Engine[††] is a plug-in to the SWRLAPI [20] that supports the execution of SWRL rules using the Drools rule engine[†††]. RDF4J[††††] provides rule-based reasoning support with SPIN (SPARQL Inferencing Notation) [21], i.e. rules in SPARQL syntax. Review and comparison of these systems is planned for future work.

3. A Performance Evaluation Study of Rule-Based Inference Engines

3.1 Overview

Performance is an important criterion for choosing an inference engine for an application. One requirement of an inference engine for the Semantic Web data is the ability to process a large amount of data, i.e. high scalability and efficiency. In addition, the performance should be evaluated according to different application requirements. Specifically, each application may be different in terms of required expressive power, a dependency of rules, and data sizes.

There have been comparative studies on the performance of rule-based inference engines for the Web scale. OpenRuleBench [22] is one of the most comprehensive studies on developing a benchmark for comparing and analyzing the performance of these systems. The study involved evaluation of broad-ranged system technologies including prolog-based, deductive databases, production rules, triple engines, and general knowledge bases. The study also involved large join tests, datalog recursion, and default negation. Our study differed from the OpenRuleBench in several aspects. First, our study only investigated inference engines that were based on triple engines and were available under free software or freeware licenses. Thus, the study involved some systems that were not in the OpenRuleBench studies. Second, our study used the dataset from the Berlin SPARQL Benchmark (BSBM) [4], which simulated an e-commerce use case involving data about products, vendors, consumers and reviews about the products. The BSBM was chosen because it was closer to real-world enterprise application scenarios than several independent datasets employed in the OpenRuleBench. Third, our study applied tests on some general expressive power which were not in the OpenRuleBench study. Examples of such tests were built-in functions, and fact retraction.

3.2 Methodology

3.2.1 Dataset Description

Datasets and reasoning tasks are the main components of reasoner evaluation resources [23]. In our study, the dataset was adapted from the BSBM package which consists of dataset generators and queries mix that can be used for comparing the performance of native RDF stores. The benchmark built around an e-commerce use cases in which a set of products was offered by different vendors and consumers had posted reviews about products. The BSBM dataset consists of the following classes: product, product type, product feature, producer, vendor, review, and person.

Although the BSBM was not originally designed as a benchmark tool for inference engines, it was chosen because it could simulate real-world enterprise application scenarios, e.g., to create product recommendations for customers. In addition, the BSBM dataset is provided in the RDF data format, which simulates the Semantic Web data setting. In our tests, five different sizes of the dataset were generated and varied by the number of products: 1 K, 1.5 K, 2 K, 2.5 K, and 3 K products. The numbers of generated triples were ranged from 375 K to 1 M triples respectively.

3.2.2 Ruleset Description

Six rulesets were designed to test the performance of forward-chaining reasoning tasks, i.e., matching, action, and firing rules. The rulesets used the vocabularies from the benchmark dataset. Each ruleset is different in various categories of factors: join complexity, production operation, negation, built-in functions, and rule dependency. These factors can be briefly described as follows.

[††]https://github.com/protegeproject/swrlapi-drools-engine
[†††]http://www.drools.org/
[††††]http://rdf4j.org/

```
[(?product rdf:type bsbm:Product) (?product rdfs:label ?label) regex(?label,(.)*ac(.)*")
(?product bsbm:productFeature bsbm-inst:ProductFeature1) (?product
bsbm:productPropertyNumeric1, ?property) ge(?property, 1000)
(?offer rdf:type bsbm:Offer)(?offer bsbm:product ?product) (?offer bsbm:price ?price) le(?price,
5000) (?offer bsbm:vendor dataFromVendor1:Vendor1)
->(:userSuggestedProducts :userPreferenceProduct ?product)]
```

Fig. 1 An example rule syntax of ruleset 1

```
[(?product rdf:type bsbm:Product)(?product bsbm:productFeature bsbm-inst:ProductFeature1)
-> (:SummerProducts :product ?product)]
[(?product rdf:type bsbm:Product)(?product bsbm:productFeature bsbm-inst:ProductFeature2)
-> (:SummerProducts :product ?product)]
[(?product rdf:type bsbm:Product)(?product bsbm:productFeature bsbm-inst:ProductFeature3)
-> (:SummerProducts :product ?product)]
[(?product rdf:type bsbm:Product)(?product bsbm:productFeature bsbm-inst:ProductFeature4)
-> (:SummerProducts :product ?product)]
[(?product rdf:type bsbm:Product)(?product bsbm:productFeature bsbm-inst:ProductFeature5)
-> (:SummerProducts :product ?product)]
[(:SummerProducts :product ?product) (:CurrentTime :month ?month) le(?month, 8)
ge(?month, 4) -> (:userSuggestedProducts :seasonPreferenceProducts ?product)]
```

Fig. 2 An example rule syntax of ruleset 2

- **Join complexity**: The uses of joins are classified into three categories: low, medium, and high complexity. Low join complexity is defined as when fewer than three atoms are involved in joins. Medium complexity is defined as when between three and seven atoms are involved. High complexity is defined as when more than seven atoms are involved.
- **Operation**: Operations are defined as whether assertion or retraction is involved in the consequence part.
- **Negation**: Negation is defined as whether negation is used in the antecedent part.
- **Built-in functions:** Built-in function is defined as whether string or math functions, or comparators are used.
- **Rule dependency:** Rule dependency is defined as whether the result of a rule will result in the firing of another rule or firing the same rule, i.e. recursion.

Six rulesets were created to emulate the business logics of recommending products to users and data modification. Using these rulesets, an e-commerce recommender system can generate product recommendation results for the user based on the three information types: user preferences, season preferences, and product-rating preferences. In addition, the system modified the delivery days and the price of products sold by a vendor based on some defined criteria. Four rulesets consisted of one rule per ruleset, and two rulesets consisted of more than one rule per ruleset. Table 2 provides a summary of characteristics for each ruleset. The details of each ruleset are provided as follows.

Table 2 Summary of characteristics of the rulesets

Ruleset	**1**	**2**	**3**	**4**	**5**	**6**
Join complexity	high	Low	medium	medium	high	low
Operation						
Assertion	✓	✓	✓	✓	✓	✓
Retraction	-	-	-	✓	-	-
Negation	-	-	-	✓	✓	-
Built-in Function						
Math	✓	✓	✓	✓	✓	-
String	✓	-	-	-	✓	-
Rule dependency	-	-	-	-	-	recursion
Number of Rules	1	24	1	1	1	2

Ruleset 1: Recommend products based on user preferences. The user preferences included text in product label, product features, price, and vendor of products. This ruleset involved complex join (eight atoms) and function symbols. An example of the rule syntax in the Jena rule format is shown in Fig. 1.

Ruleset 2: Recommend products based on season preferences. Products were matched with each season by five product features. The example showed the partial rules of summer season only. This ruleset involved simple join (two atoms) of a large number of rules (24 rules). An example of the rule syntax in the Jena rule format is shown in Fig. 2.

Ruleset 3: Recommend products based on product-rating preferences. This ruleset involved intermediate join (three

```
[(?product rdf:type bsbm:Product)(?review bsbm:reviewFor ?product)
(?review bsbm:rating4 ?rating4) ge(?rating4, 8)
-> (:userSuggestedProducts :popularProduct ?product)]
```

Fig. 3 An example rule syntax of ruleset 3

```
[(?offer rdf:type bsbm:Offer)(?offer bsbm:vendor dataFromVendor5:Vendor5) (?offer bsbm:deliveryDays ?days)
noValue(?offer :firedFor bsbm:deliveryDays) sum(?days, 5, ?newDays) (?offer bsbm:price ?price)
noValue(?offer :firedFor bsbm:price) product(?price, 0.9,?newPrice)
-> remove(2) remove(5) (?offer bsbm:deliveryDays ?newDays) (?offer bsbm:price ?newPrice) (?offer :firedFor
bsbm:deliveryDays) (?offer :firedFor bsbm:price)]
```

Fig. 4 An example rule syntax of ruleset 4

```
noValue(?product bsbm:productFeature bsbm-inst:ProductFeature5)
noValue(?product bsbm:productFeature bsbm-inst:ProductFeature10)
```

Fig. 5 An example rule syntax of ruleset 5

```
[(?reviewFromA bsbm:reviewFor ?product) (?reviewFromB bsbm:reviewFor ?product)
(?reviewFromA rev:reviewer ?A) (?reviewFromB rev:reviewer ?B)
(?offer bsbm:product ?product) (?offer bsbm:price ?price) lessThan(?price, 30)
(?offer bsbm:deliveryDays ?days) lessThan(?days, 3) -> (?A :connects ?B)]
```

Fig. 6 An example rule syntax for the base case data generator of ruleset 6

```
[(?A :connects ?B) (?B :connects ?C) -> (?A :connects ?C)]
```

Fig. 7 An example rule syntax of ruleset 6

atoms). An example of the rule syntax in the Jena rule format is shown in Fig. 3.

Ruleset 4: Extend the delivery days of products sold by 'Vendor5' by five days, and decrease the price by ten percent. Product data modification was written as a combination of two operations: retraction and assertion. The assertion of *:firedFor* predicate was the flag of modified triples, and the condition of the form *noValue(?offer :firedFor <predicate>)* was added to prevent them from being matched again (firing itself). This ruleset involved intermediate join (four atoms), negation, and retraction. An example of the rule syntax in the Jena rule format is shown in Fig. 4.

Ruleset 5: This ruleset extended ruleset 1 by adding two negation atoms to exclude products having two specific product features. Only added negation atoms were shown in Fig. 5. This ruleset involved complex join and negation.

Ruleset 6: This ruleset involved a large number of recursions. The base cases of recursion were connections of reviewers who wrote reviews of the same product. The recursion was transitive closure based on these relations (e.g. two groups of people could connect to each other via some people who were in both groups). The base cases of the recursion were generated by the base case data generator. The results were stored as a pre-processing step so that only recursion was involved in this test. The conditions of price and delivery days of products were used to limit the number of involved reviewers. An example rule syntax for the base case data generator is shown in Fig. 6. An example of the rule syntax of ruleset 6 is shown in Fig. 7.

Mixed ruleset: Combination of rulesets 1, 2, and 3.

3.2.3 Experiment Settings

All tests were performed on a machine with the following specifications: Intel Core i5 430 M CPU, and 8 GB main memory size. The machine was running on Windows 8 and JDK version 1.6 with maximum heap size 6 GB. In each test, we warmed up the inference engines by loading data and perform reasoning until the execution time was stable. Then we measured the average system response time of three-time executions. The following provides the version numbers and configurations of the tested inference engines.

Table 3 Execution time of data loading (in seconds)

No. Products	EYE	Jena	BaseVISor
1,000	72.33	1.78	30.85
1,500	107.23	3.19	38.72
2,000	135.95	4.00	50.38
2,500	164.45	6.10	76.31
3,000	189.55	6.45	82.68

1. Apache Jena Inference Engine from Jena framework version 2.11.0 using forward chaining strategy with RDF datasets in Turtle format, and rulesets in Jena's own rule format
2. Euler YAP Engine (EYE) version 2013-05 with RDF datasets in Turtle format, and rulesets in Notation3 format
3. BaseVISor version 2.0, with configuration "optimizationLevel = high" with RDF datasets and rulesets in BaseVISor's own .bvr format

The performance metrics used were load time and reasoning time. The load time was the time the system used in loading the dataset. The reasoning time was measured from after the data was loaded into the main memory until the inference phase had finished. The RDFS/OWL inference rules were not applied to the data because they did not affect the inference results of the user-defined rulesets and thus were not involved in the reasoning time.

3.3 Results

3.3.1 Load Time

Table 3 shows the execution time of loading dataset into the inference engines. Comparing the load time performance, Jena inference engine was the fastest for all sizes of datasets, followed by BaseVISor and EYE. At 3,000 products, Jena was approximately factor of 12.8 faster than BaseVISor and factor of 29.3 faster than EYE. The more load time indicated the more time that the systems used in managing indexing of the RDF data.

3.3.2 Reasoning Time

Figures 8–14 show the result comparison charts for each ruleset. Euler YAP Engine was not included in the results of rulesets 4 and 5 because its supported rule language, Notation3, did not support non-monotonic negation and retraction.

For rulesets 1 and 5 which involved complex join, BaseVISor yielded the best performance with a low growth rate. For ruleset 1, Euler YAP Engine showed the second best performance with a similar growth rate to BaseVISor. Jena's did not perform well in both rulesets with a higher growth rate. The execution times of the ruleset 5 for Jena's were slightly lower than the ruleset 1. The lower execution time might be because of the exclusion of some results by

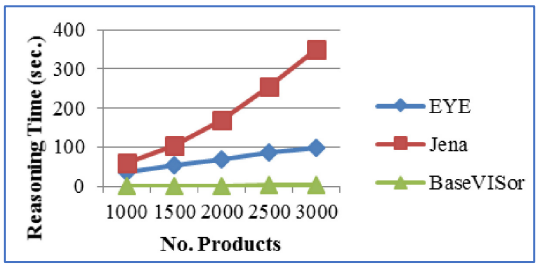

Fig. 8 Result comparison of ruleset 1

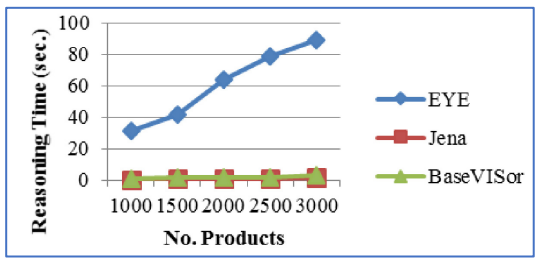

Fig. 9 Result comparison of ruleset 2

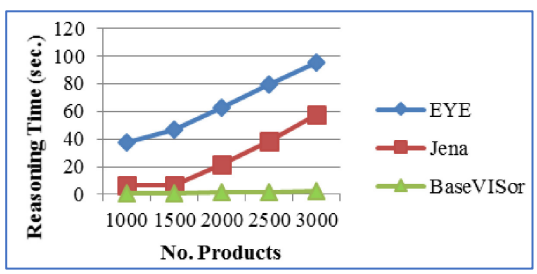

Fig. 10 Result comparison of ruleset 3

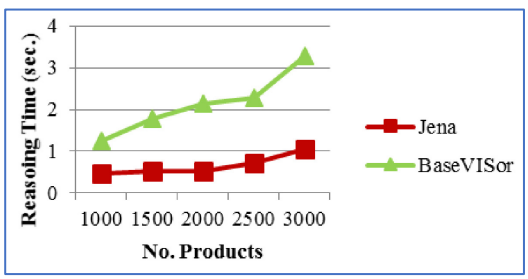

Fig. 11 Result comparison of ruleset 4

negation, which reduced the size of input to the join operations. The result of the mixed rulesets (1+2+3) also showed the performance in the similar growth rates as the ruleset 1 for all systems. These results have indicated that join complexity has the most effects on the performance.

For the ruleset 2 which involved simple join of several rules, the result of Jena inference engine was significantly better than the result of ruleset 1 and was slightly better than BaseVISor's. Euler YAP Engine's showed relatively similar performance and trend as the result of ruleset 1.

For ruleset 3 which involved intermediate join, Base-

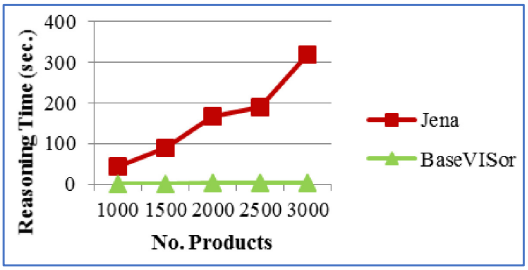

Fig. 12 Result comparison of ruleset 5

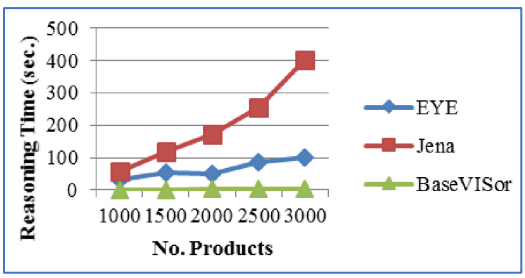

Fig. 13 Result comparison of mixed ruleset

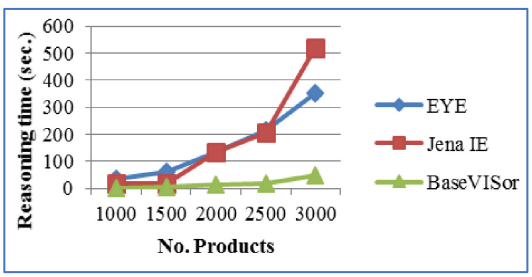

Fig. 14 Result comparison of ruleset 6

VISor's had the best performance with a lower growth rate. Jena's showed the second best performance with a higher growth rate. Euler YAP Engine yielded the third best performance with a similar growth rate as Jena's. The result of Euler YAP Engine also had similar performance and trend as the result of ruleset 1.

For ruleset 4, which involved intermediate join, retraction and negation, Jena showed the best performance with a lower growth rate. BaseVISor's had lower performance with higher growth rate than Jena inference engine. The execution times of the ruleset 4 for Jena's were lower than that of the ruleset 3. This might be because some results were excluded from joining by negation and retraction.

For ruleset 6, which involved recursion of simple join, BaseVISor showed the best performance with a lower growth rate. Euler YAP Engine had the second best performance with a higher growth rate. Jena's yielded the third best performance with a similar growth rate as Euler YAP Engine at between 1,000 to 2,500 products but the growth highly increased at 3,000 products.

Based on the results, we concluded that BaseVISor's had the best overall performance with a low growth rate. Except for the ruleset 4, BaseVISor's performed the best among the studied inference engines. The results showed that Jena was extremely fast when simple join was involved (ruleset 2), but did not yield an outstanding performance for complex join (rulesets 1, 5, mixed) or recursion (ruleset 6). Jena's also had performance better than other systems for retraction and negation when complex join was not involved (ruleset 4). Euler YAP Engine had a longer loading time. Its results for all of our tests were surprisingly in the similar value ranges and growth rates (except for ruleset 6).

We made some observations based on the results as follows. Both BaseVISor and Jena's are based on RETE inference algorithm. However, BaseVISor claimed that its implementation was optimized for RDF triples by using a simple data structure for its facts (i.e., triples) rather than arbitrary list structures, which permitted greatly enhanced efficiency in pattern matching of RETE network [18]. In contrast to Jena's, the performance of BaseVISor was marginally affected by the complexity of join, which indicated the effectiveness of its join optimization technique. Euler YAP Engine performance was also marginally affected by the complexity of join for a similar reason, although its forward-chaining reasoning algorithm was generally less effective than BaseVISor's RETE-based algorithm.

4. Conclusions

In this article, we reviewed and compared some freely-available rule-based reasoners designed for the Semantic Web data. One of the main purposes is to identify some criteria that can be used to compare rule-based reasoners for the Semantic Web. We compared some key features of three rule-based reasoners: Jena's, EYE, and BaseVISor, based on these criteria. Our work also conducted a performance evaluation study of these rule-based reasoners based on the Berlin SPARQL Benchmark. The study result shows the efficiency of the systems in performing reasoning tasks over different data sizes and rules involving various rule properties including join, reasoning operation, negation, built-in function, and rule dependency. Our experimental result analysis can be used in guiding researchers and developers in the Semantic Web field to better choose rule-based reasoners that match their application requirements. Our planned future work will consider additional benchmark, such as the benchmark of additional reasoning strategy, i.e., backward reasoning, and comparative study of additional inference engines.

Acknowledgements

The financial support from Young Scientist and Technologist Program, NSTDA is gratefully acknowledged.

References

[1] W3C, "Semantic Web," [Online] Available: https://www.w3.org/standards/semanticweb/ [Accessed: 24-May-2017].

[2] S. Singh and R. Karwayun, "A comparative study of inference engines," Proc. 7th International Conference on Information Technology: New Generations, pp.53–57, 2010.
[3] W3C, "Category:Reasoner - Semantic Web Standards," [Online] Available: http://www.w3.org/2001/sw/wiki/Category:Reasoner [Accessed: 24-May-2017].
[4] C. Bizer and A. Schultz, "The Berlin SPARQL benchmark," Int. J. Semant. Web Inf. Syst., vol.5, no.2, pp.1–24, April 2009.
[5] J.C. Giarratano and G. Riley, Expert Systems: Principles and Programming, 2nd ed., PWS Publishing, Boston, MA, USA, 1994.
[6] P.J. Hayes and P.F. Patel-Schneider, "RDF 1.1 Semantics," [Online] Available: http://www.w3.org/TR/rdf11-mt/ [Accessed: 24-May-2017].
[7] T. Rattanasawad, K.R. Saikeaw, M. Buranarach, and T. Supnithi, "A review and comparison of rule languages and rule-based inference engines for the Semantic Web," Proc. 2013 International Computer Science and Engineering Conference - Workshop on Ontology and Semantic Web for Big Data, pp.1–6, 2013.
[8] M. Kifer and H. Boley, "RIF Overview (Second Edition)," [Online] Available: http://www.w3.org/TR/rif-overview/ [Accessed: 24-May-2017].
[9] T. Berners-Lee and D. Connolly, "Notation3 (N3): A readable RDF syntax," [Online] Available: https://www.w3.org/TeamSubmission/n3 [Accessed: 24-May-2017].
[10] Apache Software Foundation, "Apache Jena - Reasoners and rule engines: Jena inference support," [Online] Available: http://jena.apache.org/documentation/inference [Accessed: 24-May-2017].
[11] Apache Software Foundation, "Apache Jena," [Online] Available: http://jena.apache.org/ [Accessed: 24-May-2017].
[12] C.L. Forgy, "Rete: A fast algorithm for the many pattern/many object pattern match problem," Artif. Intell., vol.19, no.1, pp.17–37, Sept. 1982.
[13] J. De Roo, "Euler Yet Another Proof Engine – EYE," [Online] Available: http://eulersharp.sourceforge.net/ [Accessed: 24-May-2017].
[14] R. Verborgh and J. De Roo, "Drawing conclusions from Linked data on the Web: The EYE reasoner," IEEE Software, vol.32, no.3, pp.23–27, April 2015.
[15] V.S. Costa, R. Rocha, and L. Damas, "The YAP Prolog System," Theory and Practice of Logic Programming - Prolog Systems, vol.12, no.1-2, Jan. 2012.
[16] H. Boley and M. Kifer, "RIF Basic Logic Dialect (Second Edition)," [Online] Available: http://www.w3.org/TR/rif-bld/ [Accessed: 24-May-2017].
[17] VIStology Inc., "BaseVISor 2.0," [Online] Available: http://vistology.com/products/ [Accessed: 24-May-2017].
[18] C.J. Matheus, K. Baclawski, and M.M. Kokar, "BaseVISor: A triples-based inference engine outfitted to process RuleML and R-Entailment rules," Proc. 2nd International Conference on Rules and Rule Languages for the Semantic Web, pp.67–74, 2006.
[19] E. Sirin, B. Parsia, B.C. Grau, A. Kalyanpur, and Y. Katz, "Pellet: A practical OWL-DL reasoner," Web Semant. Sci. Serv. Agents World Wide Web, vol.5, no.2, pp.51–53, June 2007.
[20] M. O'Connor, H. Knublauch, S. Tu, B. Grosof, M. Dean, W. Grosso, and M. Musen, "Supporting rule system interoperability on the Semantic Web with SWRL," The Semantic Web – ISWC 2005: 4th International Semantic Web Conference, pp.974–986, 2005.
[21] H. Knublauch, J.A. Hendler, and K. Idehen, "SPIN - Overview and Motivation, W3C Member Submission," [Online] Available: http://www.w3.org/Submission/spin-overview [Accessed: 24-May-2017].
[22] S. Liang, P. Fodor, H. Wan, and M. Kifer, "OpenRuleBench: An analysis of the performance of rule engines," Proc. 18th International Conference on World Wide Web, pp.601–610, 2009.
[23] B. Parsia, N. Matentzoglu, R.S. Gonçalves, B. Glimm, and A. Steigmiller, "The OWL Reasoner Evaluation (ORE) 2015 Resources," The Semantic Web – ISWC 2016: 15th International Semantic Web Conference, pp.159–167, 2016.

Thanyalak Rattanasawad received the B.Eng. degree in Computer Engineering with First-class Honors from Khon Kaen University in 2014. She is a co-researcher at the Language and Semantic Technology Lab at NECTEC in Thailand.

Marut Buranarach received the B.Eng. degree in Engineering from King Mongkut's Institute of Technology Lardkrabang in 1995. He received the M.S. and Ph.D. degrees in Information Science from the University of Pittsburgh in 1998 and 2004, respectively. He is currently a senior researcher at the Language and Semantic Technology Lab at NECTEC in Thailand.

Kanda Runapongsa Saikaew received the B.S. degree with University and College Honors in Electrical and Computer Engineering from Carnegie Mellon University, USA, in 1997. She received the M.S. and Ph.D. degrees in Computer Science and Engineering from the University of Michigan at Ann Arbor, USA, in 1999 and 2003 respectively. She is currently an associate professor of Computer Engineering department at Khon Kaen University, Thailand.

Thepchai Supnithi received the B.S. degree in Mathematics from Chulalongkorn University in 1992. He received the M.S. and Ph.D. degrees in Engineering from the Osaka University in 1997 and 2001, respectively. He is currently the head of the Language and Semantic Technology Lab at NECTEC in Thailand.