2010 Seventh International Conference on Information Technology

A Comparative Study Of Inference Engines

Swapna Singh, Ragini Karwayun
U.P.T.U.Lucknow,
Inderprastha Engineering College,
Ghaziabad, India
singhswapna@yahoo.com, raginik22@gmail.com

Abstract : **Understanding and using the data and knowledge encoded in semantic web documents requires an inference engine. Inference engines, also called reasoners, are software applications that derive new facts or associations from existing information. Inference and inference rules allow for deriving new data from data that is already known. Thus, new pieces of knowledge can be added based on previous ones. By creating a model of the information and relationships, we enable reasoners to draw logical conclusions based on the model. The use of inference engines in the semantic web allows applications to inquire why a particular conclusion has been reached, i.e. semantic applications can give proof of their conclusions. Proof traces or explains the steps involved in logical reasoning. This paper is a survey and study work, which presents a comparison of different types of inference engines in context of semantic web. It will enable to differentiate among different types of inference engines which may be beneficial to realize the various proposed prototype systems with different ideas and views on what an inference engine for semantic web should do.**

Key Words: Inference Engine, Knowledge Representation, Logical Reasoning, Semantic Web.

I. INTRODUCTION

The inference engine can be described as a form of finite state machine with a cycle consisting of three action states: match rules, select rules, and execute rules. In the first state, match rules, the inference engine finds all of the rules that are satisfied by the current contents of the data store. When rules are in the typical condition-action form, this means testing the conditions against the working memory. The rule matchings that are found are all candidates for execution: they are collectively referred to as the conflict set. Note that the same rule may appear several times in the conflict set if it matches different subsets of data items. The pair of a rule and a subset of matching data items is called an instantiation of the rule. The inference engine then passes along the conflict set to the second state, select rules. In this state, the inference engine applies some selection strategy to determine which rules will actually be executed. The selection strategy can be hard-coded into the engine or may be specified as part of the model. In the larger context of Artificial Intelligence, these selection strategies as often referred to as heuristics. Finally the selected instantiations are passed over to the third state, execute rules. The inference engine executes or fires the selected rules, with the instantiation's data items as parameters. Usually the actions in the right-hand side of a rule change the data store, but they may also trigger further processing outside of the inference engine (interacting with users through a graphical user interface or calling local or remote programs, for instance). Since the data store is usually updated by firing rules, a different set of rules will match during the next cycle after these actions are performed. The inference engine then cycles back to the first state and is ready to start over again. This control mechanism is referred to as the recognize-act cycle. The inference engine stops either on a given number of cycles, controlled by the operator, or on a quiescent state of the data store when no rules match the data.

A semantic reasoner, reasoning engine, rules engine, or simply a reasoner, is a piece of software able to infer logical consequences from a set of asserted facts or axioms. The notion of a semantic reasoner generalizes that of an inference engine, by providing a richer set of mechanisms to work with. The inference rules are commonly specified by means of an ontology language, and often a description language. In logic, a rule of inference (also called a transformation rule) is a function from sets of formulae to formulae. The argument is called the premise set (or simply premises) and the value the conclusion. They can also be viewed as relations holding between premises and conclusions, whereby the conclusion is said to be inferable (or derivable or deducible) from the premises. If the premise set is empty, then the conclusion is said to be a theorem or axiom of the logic. Many reasoners use first-order predicate logic to perform reasoning; inference commonly proceeds by forward chaining and backward chaining.

There are two types of inference engines : forward chaining and backward chaining. Forward chaining starts with the available data and uses inference rules to extract more data (from an end user for example) until a goal is reached. An inference engine using forward chaining searches the inference rules until it finds one where the antecedent (If clause) is known to be true. When found it can conclude, or infer, the consequent (Then clause), resulting in the addition of new information to its data. As the data determines which rules are selected and used, this method is also called data-driven, in contrast to goal-driven backward chaining inference.

Backward chaining starts with a list of goals (or a hypothesis) and works backwards from the consequent to the antecedent to see if there is data available that will support

978-0-7695-3984-3/10 $26.00 © 2010 IEEE
DOI 10.1109/ITNG.2010.198

any of these consequents. An inference engine using backward chaining would search the inference rules until it finds one which has a consequent (Then clause) that matches a desired goal. If the antecedent (If clause) of that rule is not known to be true, then it is added to the list of goals (in order for your goal to be confirmed you must also provide data that confirms this new rule). As the list of goals determines which rules are selected and used, this method is also called goal-driven.

In the next section, we outline the functional description of various inference engines selected for our comparative study. Section III gives an exhaustive comparative chart for the selected inference engines measured on different performance metrics. Section IV concludes this paper with a discussion of the scope and limitations of the comparative study we have performed.

II. INFERENCE ENGINES

In our comparative analysis we have studied following inference engines

A. Jess

Jess[7] is a rule engine and scripting environment written entirely in Sun's Java language by Ernest Friedman-Hill at Sandia National Laboratories in Livermore, CA. Using Jess, one can build Java software that has the capacity to "reason" using knowledge you supply in the form of declarative rules. Jess is small, light, and one of the fastest rule engines available. Its powerful scripting language gives you access to all of Java's Application Programming Interfaces. The reference implementation of Java Specification Request 94 is a driver for Jess; with it, you can connect Jess to Java software using the vendor-independent JSR 94 API.

B. Hoolet

Hoolet[8] is an implementation of an Web Ontology Language – Description Logic (OWL-DL) reasoner that uses a first order logic. The ontology is translated to collection of axioms (in an obvious way based on the OWL semantics) and this collection of axioms is then given to a first order logic for consistency checking. Hoolet is implemented using the WonderWeb OWL API for parsing and processing OWL, and the Vampire logic for reasoning purposes. Hoolet has been extended to handle rules through the addition of a parser for Resource Description Framework rule syntax and an extension of the translator to handle rules. This is again a straightforward translation based on the semantics of Semantic Web Rule Language (SWRL) rules.

C. Pellet

Pellet[9] is an open source reasoner for OWL 2 DL in Java. It provides standard and cutting-edge reasoning services for OWL ontologies. For semantically-enabled applications that need to represent and reason about information using OWL, Pellet is the leading choice for systems where sound-and-complete OWL DL reasoning is essential. For example, OwlSight is a lightweight, ontology browser for OWL that uses Google Web Toolkit (GWT), and Pronto is a probabilistic DL reasoner integrated with Pellet. Pellet is a core component of ontology-based data management applications. It also incorporates various optimization techniques, including novel optimizations for nominals, conjunctive query answering, and incremental reasoning.

D. SHER

Scalable Highly Expressive Reasoner[10] (SHER) is a breakthrough technology that provides ontology analytics over highly expressive ontologies (OWL-DL without nominals). SHER does not do any inferencing on load; hence it deals better with quickly changing data (the downside is, of course, that reasoning is performed at query time). The tool can reason on approximately seven million triples in seconds, and it scales to data sets with 60 million triples, responding to queries in minutes. It has been used to semantically index 300 million triples from medical literature. SHER tolerates logical inconsistencies in the data, and it can quickly point you to these inconsistencies in the data and help you clean up inconsistencies before issuing semantic queries. The tool explains (or justifies) why a particular result set is an answer to the query; this explanation is useful for validation by domain experts.

E. KAON2

KAON2[11] is a successor to the KAON project (often referred to as KAON1). The main difference to KAON1 is the supported ontology language: KAON1 used a proprietary extension of RDFS, whereas KAON2 is based on OWL-DL and Frame Logic (F-Logic). Please note that KAON2 is a completely new system, and is not backward-compatible with KAON1. KAON2 is an infrastructure for managing OWL-DL, SWRL, and F-Logic ontologies. KAON2 provides the following features: An API for programmatic management of OWL-DL, SWRL, and F-Logic ontologies, A stand-alone server providing access to ontologies in a distributed manner using Remote Method Invocation, An inference engine for answering conjunctive queries (expressed using SPARQL Protocol and RDF Query Language-SPARQL syntax), A DL Implementation Group (DIG) interface, allowing access from tools such as Protege, A module for extracting ontology instances from relational databases. KAON2 supports answering conjunctive queries, although without true non-distinguished variables. This means that all variables in a query are bound to individuals explicitly occurring in the knowledge base, even if they are not returned as part of the query answer.

F. RacerPro

RacerPro[12] is an OWL Reasoner and Inference Server for the Semantic Web. RACER stands for Renamed ABox and Concept Expression Reasoner. With RacerPro one can implement industrial strength projects based on the W3C standards RDF and OWL, and it is an ideal tool for research and development. RacerPro can process OWL Lite as well as OWL DL documents (knowledge bases). RacerPro is a knowledge representation system that implements a highly optimized tableau calculus for a very expressive description

logic. RacerPro also provides facilities for algebraic reasoning including concrete domains for dealing with: min/max restrictions over the integers, linear polynomial equations over the reals or cardinals with order relations, equalities and inequalities of strings.

G. Jena

Jena[13] is an open source Semantic Web framework for Java. It provides an API to extract data from and write to RDF graphs. The graphs are represented as an abstract "model". A model can be sourced with data from files, databases, URLs or a combination of these. A Model can also be queried through SPARQL and updated through SPARUL -SPARQL/Update is an extension to the SPARQL query language, that provides the ability to add, update, and delete RDF. Jena is similar to Sesame; though, unlike Sesame, Jena provides support for OWL (Web Ontology Language). The framework has various internal reasoners and also provides support for external reasoners through the DIG interface. In addition the Pellet reasoner (an open source Java OWL-DL reasoner) can be set up to work in Jena without using the DIG interface. This allows for improved speed and overcomes some of the limitations in the DIG protocol.

H. FaCT

FaCT (Fast Classification of Terminologies) is a Description Logic (DL) classifier that can also be used for modal logic satisfiability testing. The FaCT system includes two reasoners, one for the logic SHF (ALC augmented with transitive roles, functional roles and a role hierarchy) and the other for the logic SHIQ (SHF augmented with inverse roles and qualified number restrictions), both of which use sound and complete tableaux algorithms. FaCT's most interesting features are one its expressive logic (in particular the SHIQ reasoner): SHIQ is sufficiently expressive to be used as a reasoner for the Dedicated Logic Register (DLR) logic, and hence to reason with database schemata; two its support for reasoning with arbitrary knowledge bases (i.e., those containing general concept inclusion axioms); three its optimised tableaux implementation (which has now become the standard for DL systems), and its Common Object Request Broker Architecture - CORBA based client-server architecture.

I. FaCT++

FaCT++[14] is the new generation of the well-known FaCT OWL-DL reasoner. FaCT++ uses the established FaCT algorithms, but with a different internal architecture. Additionally, FaCT++ is implementated using C++ in order to create a more efficient software tool, and to maximise portability.

J. SweetRules

SweetRules[15] is a uniquely powerful integrated set of tools for semantic web rules and ontologies, revolving around the RuleML (Rule Markup/Modeling Language) emerging standard for semantic web rules, and supporting also the closely related SWRL (Semantic Web Rule Language), along with the OWL standard for semantic web ontologies, which in turn use XML and, optionally, RDF. (SWRL rules are essentially an expressive subset of RuleML rules). SweetRules supports the powerful Situated Courteous Logic Programs extension of RuleML, including prioritized conflict handling and procedural attachments for actions and tests. SweetRules capabilities include semantics-preserving translation and interoperability between a variety of rule and ontology languages (including XSB Prolog which is a Logic Programming and Deductive Database system for Unix and Windows., Jess production rules, HP Jena-2, and IBM CommonRules), highly scaleable backward and forward inferencing, and merging of rulebases/ontologies.

K. OWLIM

OWLIM - OWLMemSchemaRepository[16] is a high-performance semantic repository developed in Java. It is packaged as a Storage and Inference Layer (SAIL) for the Sesame RDF database. OWLIM is based on Triple Reasoning and Rule Entailment Engine (TRREE) – a native RDF rule-entailment engine. The supported semantics can be configured through rule-set definition and selection. The most expressive pre-defined rule-set combines unconstrained RDFS with most of OWL Lite (as indicated on the OWL fragments map).

L. F-OWL

F-OWL[17] is the OWL inference engine that uses a Frame-based System to reason with OWL ontologies. F-OWL is accompanied by a simple OWL importer that reads an OWL ontology from a URI and extracts RDF triples out of the ontology. The extracted RDF triples are converted to format appropriate for F-OWL's frame style and fed into the F-OWL engine. It then uses flora rules defined in flora-2 language to check the consistency of the ontology and extract hidden knowledge via resolution.

M. BaseVISor

BaseVISor[18] is a forward-chaining inference engine based on a Rete network optimized for the processing of RDF triples. BaseVISor has been outfitted to process RuleML and R-Entailment rules. In the case of RuleML, n-ary predicates are automatically translated into binary predicates and refined statements that encapsulate the n-ary predicates' arguments. For R-Entailment, the R-Entailment axioms, axiomatic triples and consistency rules are imported into the engine and then used to derive all triples entailed by a base set of triples. Operation of the system will be demonstrated using sample rule sets employing RuleML and R-Entailment.

III. COMPARITIVE CHART

Search Engines	OWL-DL Entailment	Supported Expressivity For Reasoning	Reasoning Algorithm	Consistency Checking	DIG Support	Rule Support	Version	Licencing
JESS	Yes	-	Rule Based	Yes	No	Yes(SWRL)	7.1	Non Free/Closed-source
HOOLET	Yes	-	First-Order Prover	No	No	Yes(SWRL)	-	Free/Open-source
PELLET	Yes	SROIQ(D)	Tableau	Yes	Yes	Yes(SWRL-DL Safe Rules)	2.0 RC5	Free/ Open - Source& Non Free closed source
SHER	Yes	SHIN	Rule Based	Yes	Yes	Yes(SWRL-DL Safe Rules)	-	Free/Open-source
KAON2	Yes	SHIQ(D)	Resolution & Datalog	-	Yes	Yes(SWRL-DL Safe Rules)	-	Free/closed-source
RACERPRO	Yes	-	Tableau	Yes	Yes	Yes(SWRL-not fully support SWRL)	1.9.2	Non-Free/closed-source
JENA	No complete reasoner included with standard distribution	Various reasoner (incomplete for nontrivial description logics)	Rule Based	Incomplete for OWL-DL	Yes	Yes(own Rule Format)	2.5.4	Free/Open-source
FACT	Yes	SHIQ	Tableau	Yes	Yes	No	-	Free/open-source
FACT++	Yes	SROIQ(D)	Tableau	Yes	Yes	No	1.3.0	Free/open-source
SWEETRULES	No	-	Rule Based	No	No	Yes(SWRL, RuleML, Jess)	2.1	Free/open-source
OWLIM	No	R-entailment	Rule Based	No	No	Yes(own Rule Format)	2.x/3.x	Free/ Open - Source& Non Free closed source
F-OWL	Yes	-	Tableau	Yes	No	Yes(SWRL)	0.3	Free/open-source
BASEVISOR	No	R-entailment	Rule Based	Yes	No	Yes(SWRL, RuleML, Jess)	1.x	Free/ Open - Source& Non Free closed source

Note : "-" instead of N/A means that the performance of the respective inference engine under the given criteria is not clear.

IV. CONCLUSION

We have briefly described how various inference engines fair when compared on different measurement criteria. We tried to be exhaustive in our study but there may be other engines which we have not discussed. Our comparative analysis may be further helpful in selecting various technologies while using ontology based inference engines for semantic web applications and for future research works.

ACKNOWLEDGMENT

We would like to thank to our Director for providing the full cooperation, support and for helpful comments on an earlier draft of this paper. We sincerely thank Prof. R.K.Bassi, for his guidance, help and motivation. Apart from the subject, we learnt a lot from him, which we are sure will be useful in different stages of our life. Finally, this paper would not have been possible without the confidence, endurance and support of our family. Our family has always been a source of inspiration and encouragement.

REFERENCES

[1] G. Antoniou, F.van Harmelen. A semantic web primer, The MIT Press.

[2] T.Berners-Lee e.a., Semantic Web Tutorial Using N3, May 20,2003, www.w3.org/2000/10/swap/doc/

[3] J.Mayfield, T.Finin, and B.County,"Information retrieval on the semantic web: Integrating inference and retrieval," in SIGIR Workshop on the Semantic web,Toronto,Canada,2004.

[4] J.D.Heflin, Towards the semantic web: knowledge representation in a dynamic, distributed environment, Dissertation 2001

[5] G.Klyne, Nine by nine:Semantic Web Inference using Haskell, May, 2003,

[6] www.ninebynine.org/Software/swish-0.1.html

[7] http://www.jessrules.com/

[8] http://owl.man.ac.uk/hoolet/

[9] http://clarkparsia.com/pellet

[10] Julian Dolby, Achille Fokoue, Aditya Kalyanpur, Edith Schonberg, Kavitha Srinivas, "Scalable highly expressive reasoner (SHER)"

[11] http://kaon2.semanticweb.org/

[12] Volker Haarslev and Ralf Moller, "RACER: A Core Inference Engine for the Semantic Web".

[13] Michael Kifer, Georg Lausen, James Wu. "Logical Foundations of Object_Oriented and Frame_Based Languages" http://jena.sourceforge.net/inference/

[14] http://owl.man.ac.uk/factplusplus/

[15] http://sweetrules.semwebcentral.org/ruleformat.pdf

[16] Atanas Kiryakov, Damyan Ognyanov, Dimitar Manov, "OWLIM – a Pragmatic Semantic Repository for OWL".

[17] Youyong Zou, Tim Finin and Harry Chen, "F-OWL: an Inference Engine for the Semantic Web" supported by DARPA and NSF.

[18] Christopher J. Matheus, Robert Dionne, Douglas F. Parent , Kenneth Baclawski and Mieczyslaw M. Kokar "BaseVISor: A Forward-Chaining Inference Engine Optimized for RDF/OWL Triples" http://www.vistology.com/basevisor

[19] Jakub Moskal, Northeastern University, Chris Matheus, Vistology, Inc. Comparison of BaseVISor, Jena and Jess Rule Engines"

[20] Jess homepage. http://herzberg.ca.sandia.gov/jess/

[21] Michael Kifer, Georg Lausen, James Wu. "Logical Foundations of Object_Oriented and Frame_Based Languages" http://jena.sourceforge.net/inference/