Pellet: An OWL DL Reasoner

Bijan Parsia and Evren Sirin
MINDSWAP Research Group
University of Maryland, College Park, MD
bparsia@isr.umd.edu, evren@cs.umd.edu

Abstract

Reasoning capability is of crucial importance to many applications developed for the Semantic Web. Description Logics provide sound and complete reasoning algorithms that can effectively handle the DL fragment of the Web Ontology Language (OWL). However, existing DL reasoners were implemented long before OWL came into existence and lack some features that are essential for Semantic Web applications, such as reasoning with individuals, querying capabilities, nominal support, elimination of the unique name assumption and so forth. With these objectives in mind we have implemented an OWL DL reasoner and deployed it in various kinds of applications.

1 Introduction

Many applications developed for the Semantic Web require some kind of reasoning capability. Providing sound and complete reasoning services is essential for many of these applications to function properly. There are known effective reasoning algorithms for Description Logics that can effectively handle the Lite (as the DL SHIF(D)) and (nearly all of) the DL dialects of OWL (as the DL SHION(D)). Existing DL reasoners, most notably FaCT and Racer, are quite efficient but do not meet some important requirements. In general, a Semantic Web reasoner should handle individuals (provide ABox reasoning), should not make the Unique Name Assumption, should support entailment checks, should answer conjunctive ABox queries and should work with XML Schema datatypes. At this stage of Semantic Web adoption, having an open-source reasoner that tries to meet these requirements is critical. Pellet has been developed to address these issues and has become both our test bed for experiments with DL and Semantic Web reasoning as well as our standard reasoning component. While not (yet) in the performance range of Racer or FaCT, it has many features that makes it a good choice for various lighter weight situations.

2 Pellet's Features

Pellet has a number of features either driven by OWL requirements or Semantic Web issues.

Ontology analysis and repair OWL has two major dialects, OWL DL and OWL Full, with OWL DL being a subset of OWL Full. All OWL knowledge bases are encoded as RDF/XML graphs. OWL DL imposes a number of restrictions on RDF graphs, some of which are substantial (e.g., that the set of class names and individual names be disjoint) and some less so (that every item have a "type" triple). Ensuring that an RDF/XML document meets all the restrictions is a relatively difficult task for authors, and many existing OWL documents are nominally OWL Full, even though their authors intended for them to be OWL DL. Pellet incorporates a number of heuristics to detect "DLizable" OWL Full documents and "repair" them.

Datatype reasoning XML Schema has a rich set of "simple" datatypes including various numeric types (integers and floats), strings, and date/time types. It also has several mechanisms, both standard and unusual, for creating new types out of the base types. For example, it's possible to define a datatype by restricting the integers to the set of integers whose canonical representation has only 10 digits, or whose string representation matches a certain regular expression. Currently, XML Schema systems tend toward validation of documents. Pellet can test the satisfiability of conjunctions of thus constructed datatypes.

Entailment In Semantic Web, entailment is the key inference whereas the Description Logic community have focused on satisfiability and subsumption. While entailment can be reduced to satisfiability, most DL systems do not support it directly. In part to pass a large portion of the OWL test suite, we implemented entailment support in Pellet.

Conjunctive ABox query Query answering is yet another important feature for Semantic Web. We have implemented an ABox query answering module in Pellet using the so-called "rolling-up" technique [Horrocks and Tessaris, 2002]. We have devised algorithms to optimize the query answering by changing how likely candidates for variables are found and verified. Exploiting the dependencies between different variable bindings helps us to reduce the total number of satisfiability tests, thus speeding up the query significantly.

3 Pellet System

Pellet is based on tableaux algorithms developed for expressive Description Logics [Horrocks *et al.*, 2000]. It supports all the OWL DL constructs including owl:oneOf and owl:hasValue. Currently, there is no known sound, complete, decidable and effective algorithm for all of OWL DL (in particular, there is none for handling nominals with in-

verse properties and cardinality restrictions). Pellet uses a combination of existing sound and complete algorithms to provide reasoning that is sound and complete for OWL DL without nominals (i.e., SHIN(D)) and OWL DL without inverse properties (i.e., SHON(D)). We use these algorithms in combination to achieve sound but incomplete reasoning with regard to all of DL. This has proved practically useful in our current work.

Figure 1 shows the main components of Pellet reasoner. An OWL ontology is parsed into RDF triples (RDF/XML, N3 and N-Triple syntaxes are supported). Pellet validates the species of the ontology while the triples are converted to assertions and axioms in the knowledge base. If the ontology level is OWL Full because of missing type triples then Pellet uses some heuristics to repair the ontology. For example an untyped resource that has been used in the predicate position of a triple will be inferred to be a datatype property if the triple has a literal in the object position. As usual, Pellet stores the axioms about classes in the TBox component and stores the assertions about individuals in the ABox component. TBox partitioning, absorption and lazy unfolding optimizations are implemented. The tableau reasoner uses the standard tableau rules (as described above) and includes various standard optimizations such as dependency directed backjumping, semantic branching and early blocking strategies. Datatype reasoning for the built-in and derived primitive XML Schema datatypes are supported.

Pellet is implemented in pure Java and available under the MIT licence. The source files along with some minimal documentation can be downloaded from the Pellet Web page (http://www.mindswap.org/2003/pellet/).

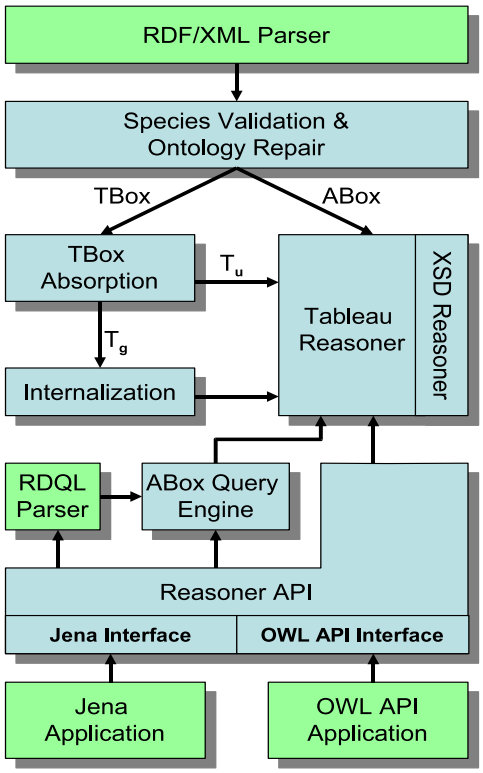

Figure 1: Architecture of Pellet reasoner

4 Applications

The capabilities of Pellet are exposed from a Java API, a command line interface, and a Web form. The Web form has been used by a number of people for species validation, consistency checking, and experimenting with OWL DL classification and entailment. Pellet provides programmatic access to the reasoning functions through two different interfaces, one for the Jena toolkit and one for the OWL API library.

Pellet is the default reasoner in Swoop, a lightweight ontology browser and editor. Pellet is used for classification, class satisfiability testing, query, and species validation and repair. Pellet is also used to find the inconsistent concept descriptions in browsed ontologies. This feature helps to locate the errors in the ontology. Generating explanations for inconsistencies would notably improve the usefulness of the reasoner in Swoop, and is being worked on as a future enhancement.

We also use Pellet for web service discovery and composition. Pellet has been incorporated as the knowledge base for a version of the SHOP2 [Nau *et al.*, 2003] planning system to evaluate the preconditions of Web Services and simulate the effects of executing these services. Pellet is also being used as a service matchmaker as a part of Fujitsu Lab of America's Task Computing Environment (TCE) [Masuoka *et al.*, 2003]. The OWL-S service descriptions are loaded to Pellet and subsumption relation between the inputs and outputs of Web Services are computed to find likely matches.

5 Future Work

We have begun experimenting with various multi-ontology/logic formalisms, such as distributed description logics and E-Connections, implementation techniques. Initial results have been both instructive and promising. We also aim to integrate the reasoner with rules to support Semantic Web Rule Language (SWRL). Other future work projects include generating explanations for concept satisfiability, querying using **K** operator and experimenting with non-monotonic reasoning using annotated logics.

6 Acknowledgements

The authors would like to thank Ron Alford, Micheal Grove and Daniel Hewitt who helped in the implementation of some modules in Pellet.

References

[Horrocks and Tessaris, 2002] Ian Horrocks and Sergio Tessaris. Querying the semantic web: a formal approach. *ISWC 2002*, pages 177–191. 2002.

[Horrocks *et al.*, 2000] I. Horrocks, U. Sattler, and S. Tobies. Practical reasoning for very expressive description logics. *Logic Journal of the IGPL*, 8(3):239–263, 2000.

[Masuoka *et al.*, 2003] Ryusuke Masuoka, Bijan Parsia, and Yannis Labrou. Task computing - the semantic web meets pervasive computing. *ISWC2003*, Florida, 2003.

[Nau *et al.*, 2003] Dana Nau, Tsz-Chiu Au, Okhtay Ilghami, Ugur Kuter, William Murdock, Dan Wu, and Fusun Yaman. SHOP2: An HTN planning system. *Journal of Artificial Intelligence Research*, December 2003.