Pellet: A Practical OWL-DL Reasoner

Evren Sirin [a] , Bijan Parsia [a] , Bernardo Cuenca Grau [a] ,
Aditya Kalyanpur [a] , Yarden Katz [a]

[a] *University of Maryland, MIND Lab, 8400 Baltimore Ave, College Park MD 20742, USA*

Abstract

In this paper, we present a brief overview of Pellet: a complete OWL-DL reasoner with acceptable to very good performance, extensive middleware, and a number of unique features. Pellet is the first sound and complete OWL-DL reasoner with extensive support for reasoning with individuals (including nominal support and conjunctive query), user-defined datatypes, and debugging support for ontologies. It implements several extensions to OWL-DL including a combination formalism for OWL-DL ontologies, a non-monotonic operator, and preliminary support for OWL/Rule hybrid reasoning. Pellet is written in Java and is open source.

Key words: Web Ontology Language, Description Logics, Tableau Theorem Proving

Introduction

Pellet started as a proof of concept system to help meet the W3C's implementation experience requirements for the Web Ontology Language (OWL). It has since become a practical and popular tool for working with OWL. Pellet has been the first reasoner to support all of OWL-DL, i.e. the Description Logic (DL) $\mathcal{SHOIN}$(D), and recently has been extended to support the new features proposed in the so-called OWL 1.1 effort [1] , i.e. the DL $\mathcal{SROIQ}$(D). OWL 1.1 extends OWL-DL with qualified cardinality restrictions, complex subproperty axioms (between a property and a property chain), local reflexiv-

Email addresses: evren@cs.umd.edu (Evren Sirin), bparsia@isr.umd.edu (Bijan Parsia), bernardo@mindlab.umd.edu (Bernardo Cuenca Grau), aditya@cs.umd.edu (Aditya Kalyanpur), yarden@umd.edu (Yarden Katz).

[1] http://owl1_1.cs.manchester.ac.uk/

ity restrictions, reflexive, irreflexive, symmetric and anti-symmetric properties, disjoint properties.

Pellet is implemented in Java and is open sourced under a liberal license. It offers a panoply of features including conjunctive query answering, rule support, $\mathcal{E}$-Connection reasoning, and axiom pinpointing, among others. To make its reasoning capabilities easily accessible to users, Pellet provides various interfaces including a command-line interface, an interactive Web form for zero-install use, DIG server implementation, and API bindings for RDF/OWL toolkits Jena and Manchester OWL-API. In this paper, we provide a brief summary of Pellet's architecture, features and special capabilities. For more details, we refer the reader to the technical report [1] and the Pellet Web site: `http://pellet.owldl.com`.

Implementation and Optimizations

System Architecture Figure 1 shows the main components of Pellet. Pellet, in its core, is a Description Logic reasoner based on tableaux algorithms. The tableaux reasoner checks the consistency of a knowledge base and all the other reasoning services are reduced to consistency checking. The reasoner is designed so that different tableaux algorithms can be plugged in. The default algorithm handles $\mathcal{SROIQ}$(D) but there are several other tableaux algorithms implemented, e.g. for non-monotonic extensions and for integration with rules.

Optimizations Pellet implements most of the state of the art optimiza-

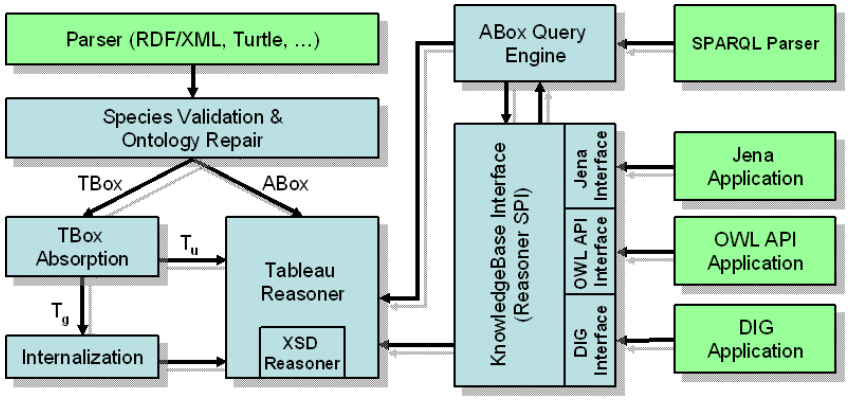

Fig. 1. Main components of the Pellet reasoner

tion techniques provided in the DL literature including *Normalization*, *Simplification*, *Absorption*, *Semantic Branching*, *Backjumping*, *Caching Satisfiability Status*, *Top-Bottom Search for Classification*, and *Model Merging* (see [2] for details about these techniques).

In addition, Pellet incorporates several novel optimizations to improve the reasoning performance in the presence of nominals (enumerated classes) and individuals. Reasoning with nominals is especially challenging since some of the existing optimizations in DL reasoners, such as *Chain Contraction*, are not applicable anymore. Moreover, in the presence of nominals, assertions about instances can affect the concept satisfiability and classification results. We have developed a suite of new optimizations to tackle this problem. Two key ones are (see [3] for the rest) are *Nominal Absorption*, a technique where axioms involving enumerations are absorbed into type assertions, and *Nominal-based Model Merging*, a technique to detect obvious non-subsumptions and non-instantiations by exploiting the fact that nominals always have a fixed interpretation in the domain. These optimizations prove sufficient to handle even the notoriously difficult Wine ontology, and, indeed, all ontologies with nominals we have encountered.

Another novel optimization technique implemented in Pellet is for incremental reasoning against dynamic knowledge bases. In many contexts (ontology editors, web portals, sensor streams), the knowledge base is in constant flux. We have developed techniques [4] to reuse the reasoning results from previous steps to process updates incrementally. Our preliminary results show up to three orders of magnitude improvement after an ABox addition or deletion.

Features and Capabilities

Conjunctive ABox Query Pellet includes a query engine that can efficiently answer conjunctive ABox queries expressed in SPARQL or RDQL. In the presence of *non-distinguished variables* in the query, the "rolling-up" technique is used to answer tree-shaped queries. Otherwise, every query atom can be answered in isolation and arbitrary

shaped queries can be handled. For such queries, the two factors affecting the query answering time are the number of atoms in the query and the order these atoms are evaluated. Pellet has two optimization techniques to deal with these cases: *Query Simplification*, finding redundant atoms in the query by using domain/range axioms, and *Query Reordering*, sorting the query atoms by utilizing a randomized sampling technique as adopted in relational databases. These techniques have been shown to be very effective in practice [5].

Datatype Reasoning Pellet uses the *type system* approach to support reasoning with datatypes. In particular, Pellet has a datatype oracle that can reason with XML Schema based datatypes. The datatype oracle can check the consistency of conjunctions of (built-in or derived) XML Schema datatypes. Pellet supports user derived types based on numeric or date/time types so, for example, numeric or date/time intervals can be defined and used as new datatypes.

Axiom Pinpointing and Debugging Axiom pinpointing is a non-standard DL inference service that provides a *justification* for any arbitrary entailment derived by a reasoner from an OWL-DL knowledge base. Given an ontology and any of its logical consequences, the axiom pinpointing service determines the premises in the KB that are sufficient for the entailment to hold. The justification is useful for understanding the output of the reasoner, which is key for many tasks, such as ontology debugging, design and evolution.

Axiom pinpointing is achieved by tracking the original source axioms from the ontology as they are modified and used throughout the tableaux algorithm. As a result, when an inconsistency is detected in the ontology, a *single* set of axioms causing the problem can be extracted. Our experiments [6] show that finding a single justification involves almost no computational overhead. Pellet can also determine *all* the justifications for an entailment by combining axiom tracing with a variant of Reiter's well-known hitting set algorithm.

Integration with Rules Formalism Pellet is coupled with a Datalog reasoner to implement the $\mathcal{AL}$-Log framework for combining DLs

with rules. $\mathcal{AL}$-Log [7] combines Datalog and DLs by allowing DL classes to be used in the body of a rule. In our implementation, we extend the $\mathcal{AL}$-Log framework to use $\mathcal{SHOIQ}$(D) (rather then the less expressive $\mathcal{ALC}$ language used in the original paper) and allow OWL datatypes and SWRL built-ins in the antecedent of Datalog rules.

Pellet also has an experimental implementation of a direct tableau algorithm for integrating DL-safe rules with $\mathcal{SHOIQ}$(D). Preliminary empirical results [8] have been encouraging and we think that the DL-safe implementation is practical for small to mid-sized ontologies especially when the full expressivity of $\mathcal{SHOIQ}$(D) is needed.

Multi-Ontology Reasoning using $\mathcal{E}$-Connections In addition to the `owl:imports` mechanism, Pellet supports a novel ontology combination technique based on $\mathcal{E}$-Connections to reason with multiple ontologies. $\mathcal{E}$-Connections are a general framework for combining several families of decidable logics and in [9] we describe tableau algorithms to combine DLs of varying expressivity. Using this technique, ontologies can be linked to each other without losing their context (in contrast to `owl:imports` which simply merges ontologies).

Non-monotonic Reasoning Non-monotonic logics have been generally successful in capturing several forms of common sense and database reasoning. A prominent family of non-monotonic formalisms are rooted in various forms of the *closed world assumption* (CWA). The DL $\mathcal{ALCK}$ [10] adds a non-monotonic **K** operator (which is a kind of necessity operator) to the DL $\mathcal{ALC}$ to provide the ability to "turn on" the CWA when needed. The reasoning support for $\mathcal{ALCK}$ language has been implemented in Pellet to answer CWA queries that use the **K** operator. We also admit a restricted use of **K** in the ontologies, in the form of an *epistemic rule*.

Conclusions and Future Work

In this paper, we have presented Pellet, an open source OWL-DL reasoner with a number of unique features. Over the years, Pellet has become a practical and popular tool because it is easily accessible through a number of interfaces, provides many standard and

extended reasoning services and exhibits a competitive performance and is open source. In the near future, we are planning to extend Pellet in several different directions. Most notably we intend to provide secondary-storage support for reasoning with large number of individuals, optimizations based on partitioning of ontologies, combinations with other logical formalisms (e.g. spatio-temporal logics), and full SWRL support.

Acknowledgments The authors would like to thank following people for their contributions to the code: Christian Halaschek-Wiener (incremental reasoning), Edna Ruckhaus ($\mathcal{AL}$-Log coupling), Vladimir Kolovski (DL-safe rules coupling), Ron Alford (TBox absorption), and Michael Grove (OWL species validation).

References

[1] E. Sirin, B. Parsia, B. C. Grau, A. Kalyanpur, Y. Katz, Pellet: A practical OWL-DL reasoner, Tech. Rep. CS 4766, University of Maryland, College Park (2005).

[2] F. Baader, D. Calvanese, D. McGuinness, D. Nardi, P. Patel-Schneider (Eds.), The Description Logics Handbook, Cambridge University Press, 2003.

[3] E. Sirin, B. Cuenca-Grau, B. Parsia, From wine to water: Optimizing description logic reasoning for nominals, in: Int. Conf. on the Principles of KR (KR-2006), 2006.

[4] C. Halaschek-Wiener, B. Parsia, E. Sirin, Description logic reasoning with syntactic updates, in: Proc. of the 5th Int. Conf. on Ontologies, Databases, and Applications of Semantics (ODBASE 2006), 2006.

[5] E. Sirin, B. Parsia, Optimizations for answering conjunctive abox queries, in: Proc. of the Int. Description Logic Workshop (DL-2006), 2006.

[6] A. Kalyanpur, B. Parsia, B. Cuenca-Grau, E. Sirin, Tableau tracing in $\mathcal{SHOIN}$, Tech. Rep. CS 4764, University of Maryland, College Park (2005).

[7] F. M. Donini, M. Lenzerini, D. Nardi, A. Schaerf, $\mathcal{AL}$-Log: Integrating datalog and description logics, Journal of Intelligent Information Systems 10 (1998) 227–252.

[8] V. Kolovski, B. Parsia, E. Sirin, Extending $\mathcal{SHOIQ}$(d) with DL-safe rules: First results, in: Proc. of the Int. Description Logic Workshop (DL-2006), 2006.

[9] B. C. Grau, B. Parsia, E.Sirin, Combining OWL ontologies using $\mathcal{E}$-connections, Journal of Web Semantics 4 (1) (2006) 40–59.

[10] F. M. Donini, M. Lenzerini, D. Nardi, A. Schaerf, W. Nutt, An epistemic operator for description logics, Artificial Intelligence 100 (1-2) (1998) 225–274.