Horned-OWL: Building ontologies at Big Data Scale

Phillip Lord[1], Jennifer Warrender[1]

[1]School of Computing, Newcastle University, Newcastle-upon-Tyne NE4 5TG, UK

Abstract

Ontologies have been widely used for representing data in biomedicine. The largest have 100,000s of concepts, and taken together all the ontologies in bioportal there are nearly 10 million classes. The computational infrastructure that we have to support these efforts though will often not scale to this size, requiring either splitting the ontology into parts or high memory computers. We have designed and built a new library, Horned-OWL, implemented in Rust that can scale to 10 million classes on a standard desktop machine, with large performance differences compared to the Java based OWL API. We believe that as well as enabling scalability, higher levels of performance can alter the way we build ontologies by making what is currently difficult, simple, and fast.

Keywords

OWL, Ontology, Software Engineering,

As the use of OWL ontologies have increased, so have their sizes. The largest ontologies now have 100,000's of terms; ontology like structures such as the NCBI taxonomy are over a million terms. Much of the infrastructure for ontology development is written in Java, as well as some Python, the latter of which operates over ontological data as text files. Neither of these languages are particularly performant either in terms of speed or their memory usage. As a result, OWL ontologies, and particularly ontology-like structures represented in OWL, are limited in their size by these practical considerations.

In this paper, we introduce Horned-OWL. This is a new library, implemented in Rust, focused on performance. While still in development, the library offers a complete OWL2-DL implementation, complete parsers for OWL-XML (OWX) format and nearly complete parser for OWL-RDF format. Our current experience suggests that it is highly performant, with order of magnitude performance in some circumstances. As such, it offers the possibility of using OWL for big data scale knowledge representation.

The Horned-OWL library is implemented in the language Rust. This language is designed as a systems programming language and has a number of attributes that support this end. While it provides a high level, abstract, functional programming idiom, the abstractions are all designed to be "zero-cost", that is they do not necessarily cost performance. This zero-cost approach also covers its memory safety system which automatically recovers used objects without the need for a runtime garbage collection system. The practical upshot of all of this, is that Rust is designed to be fast.

The Horned-OWL library follows similar design principles; it is a low-level library designed to be fast and features where runtime costs are optional, allowing the library to be tuned to the need at hand. The core `model` module, for example is a large number of **`struct`** and **`enum`**

ICBO 2021 - The 12th International Conference on Biomedical Ontologies
phillip.lord@newcastle.ac.uk (P. Lord)

declarations that together define the OWL2 data model. Unlike the OWL API, where the core data model is defined as interfaces, this is concretely defined in Horned-OWL. For example, an IRI is defined:

```
pub struct IRI(Rc<str>);
```

This is a reference counted pointer to an in memory `String`. The reference counting provides an easy mechanism to allow interning of the `String`, so each `IRI` is represented by only a single `String`. In the current implementation, Horned-OWL is hard-coded to this reference counting, but will be made generic in future versions: this means that the interning need not be used for smaller examples, therefore simplifying the code; and is also necessary to allow multi-threaded code. Equivalently, the axiom `SubClassOf` is defined:

```
pub struct SubClassOf {
    pub sup: ClassExpression,
    pub sub: ClassExpression,
}
```

This will compile to code that will use 160 bytes to represent `SubClassOf`, which is exactly twice memory needed to represent the individual `ClassExpression`. An `AnnotatedAxiom` is slightly longer (at 192 bytes). While this means that Horned-OWL is theoretically bound by the size of the computer memory, it uses that memory extremely efficiently: we predict that around six million axioms can be stored in 1GB of memory, plus the space for the IRI strings which will depend on their length. This makes it highly scalable as we show in Figure 1a.

This simplicity of implementation is also seen in the Ontology implementation. The simplest version uses an in-memory set of axioms, plus two IRIs (the ontology and version IRI). In contrast to the OWL API, this provides no indexing at all by default, while providing a pluggable mechanism should it be required. This follows the "zero-cost" idiom of Rust. For example, the OWX[1] reader does not need to perform axiom lookup during parsing, so needs no indexes, does not use them, and pays no cost for them. The RDF parser[2] meanwhile makes extensive use of indexes to allow rapid look up of declaration axioms and by logical comparison; without this, RDF parsing scales poorly[3].

While Horned-OWL has been written as a general library, we have also added a command line interface. This uses the common subcommand interface, with every command prefixed with `horned`. Currently, the available commands are rather biased toward commands which are useful for debugging ontology development libraries, but these will be expanded in future. Some of the commands include:

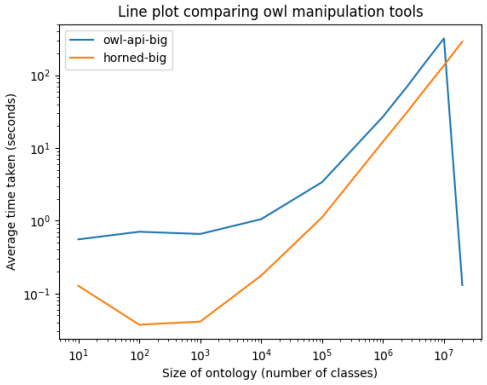
horned-big:	Generates OWL files of arbitrary size, useful for performance testing
horned-materialize:	Downloads (recursively) imported files
horned-parse:	Parses a given OWL file
horned-summary:	Prints summary statistics of an ontology

[1] https://www.w3.org/TR/owl2-xml-serialization/

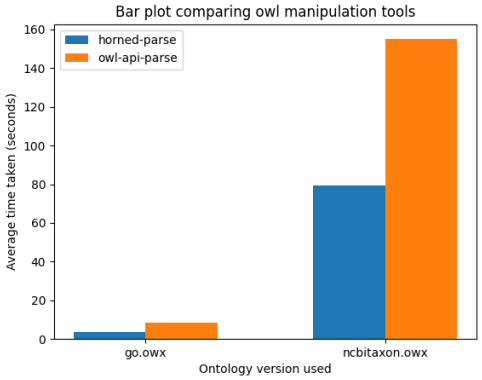
[2] https://www.w3.org/TR/owl2-mapping-to-rdf/

[3] In at least quadratic time, possibly worse; without indexes, it took several hours to load the Gene Ontology

The performant interface of Horned-OWL makes this command line quite usable; for example, the NCBI Taxonomy can be parsed in 1-2 minutes; the OWL API takes 2-3 minutes to perform an equivalent task (Figure 1b). This speed means that Horned-OWL can be used in practice to perform a series of manipulations passing output through a Unix pipe. To do the same with the OWL API would require a tool like [1], which would load the ontology once and perform multiple operations before saving[4].



(a) Generating a big ontology



(b) Parsing GO and the NCBI Taxonomy

While Horned-OWL was written as a library for ontology manipulation as well as the underpinning for the command line tools that it offers, the implementation provides a number of possibilities that we are investigating. First, while Rust is a clean, modern language, it is not easy to script with, nor particularly familiar to the ontology community. Therefore, we are currently experimenting with using it as the backend to a Python library. The heavy computational work will be performed in Rust, linked together using a Python script. Second, the Ontology Web Language despite its name is currently not usable on the web. Rust supports WebAssembly which means that it can be run within a web browser; therefore, it will provide future implementation options to make OWL available in this environment.

Acknowledgments

Thanks to James Overton and Janna Hastings for working on a Python interface to Horned-OWL.

References

[1] P. Lord, The Semantic Web takes Wing: Programming Ontologies with Tawny-OWL, in: Proceedings of the 10th International Workshop on OWL: Experiences and Directions (OWLED 2013) co-located with 10th Extended Semantic Web Conference (ESWC 2013), 2013. URL: http://ceur-ws.org/Vol-1080/, workshop proceedings CEUR-WS.org ISSN:1613-0073.

[4] Or a lot of patience. And a lot of patience while debugging the pipeline