OWL2Gen: Towards a Configurable Ontology Generator for Benchmarking

Gunjan Singh[1], Ashwat Kumar[1], Sumit Bhatia[2] and Raghava Mutharaju[1]

[1]*Knowledgeable Computing and Reasoning Lab, IIIT-Delhi, India*
[2]*Adobe Research, New Delhi, India*

Abstract
Recent advancements in OWL 2 reasoners have significantly enhanced their capabilities. However, scaling challenges persist, especially with expressive profiles such as OWL 2 DL. Effective benchmarking is essential to address these challenges. We discuss our effort towards establishing a configurable ontology generator, OWL2Gen, that empowers users to custom-build benchmark ontologies by specifying the types of axioms they need and their respective counts. With a user-friendly interface that lists all OWL 2 constructs, OWL2Gen offers an adaptable approach to ontology generation, facilitating more detailed and nuanced performance evaluations. The code and documentation are available under the Apache 2.0 License at https://github.com/kracr/owl2gen.

Keywords
OWL 2, Ontology Generator, Ontology Reasoner, Benchmarking

1. Introduction

In the past decade, there has been remarkable progress in the development of reasoners that support expressive ontology languages such as OWL 2. Despite the advancements, OWL 2 reasoners still struggle to scale well, especially for expressive language profiles like OWL 2 DL [1]. To build high-quality reasoners, developers need to identify and improve performance bottlenecks in their existing systems. A reasoner benchmark aids developers in evaluating their system's performance, addressing limitations, and paving the way for further research to enhance performance and functionality. In particular, a reasoner needs to be evaluated from several aspects, such as (a) *Coverage*, i.e., support for different OWL 2 language constructs and their combinations, as well as support for various OWL 2 profiles (EL, QL, and RL) [2]; (b) *Scalability*, i.e., the ability to handle large and expressive ontologies; (c) *Performance*, i.e., evaluation under constraints such as reasoning time and memory consumption.

Current benchmarks, such as LUBM [3], UOBM [4], and OWL2Bench [5], offer a certain degree of flexibility but remain limited in scope. They allow users to assess reasoner performance by generating ABoxes of varying sizes while the TBox remains fixed. OntoBench [6] provides a web interface for selecting OWL 2 constructs but lacks comprehensive scalability and performance

ER2024: Companion Proceedings of the 43rd International Conference on Conceptual Modeling: ER Forum, Special Topics, Posters and Demos, October 28-31, 2024, Pittsburg, Pennsylvania, USA
gunjans@iiitd.ac.in (G. Singh); ashwat16023@iiitd.ac.in (A. Kumar); sumit.bhatia@adobe.com (S. Bhatia); raghava.mutharaju@iiitd.ac.in (R. Mutharaju)

evaluation. PyGraft [7] generates domain-agnostic ontologies but does not cover all OWL 2 constructs.

To address these limitations, we present our configurable Ontology Generator, OWL2Gen. Inspired by OntoBench, we provide an interface that allows users to select and specify both the choice of constructs and their individual counts, facilitating the generation of custom-built ontologies. It leverages existing OWL 2 DL TBoxes to generate ontologies, providing a practical evaluation scenario. OWL2Gen's user-friendly interface simplifies ontology creation, making it accessible to users with varying levels of familiarity with OWL.

In the following sections, we describe OWL2Gen's methodology and outline our plans for future enhancements, aiming to expand its capabilities for comprehensive and enhanced benchmarking of OWL 2 reasoners.

2. OWL2Gen

OWL2Gen is a user-friendly application designed for flexible and configurable ontology generation. The frontend, detailed in Section 2.1, is developed using HTML, CSS, and JavaScript, and it interacts with a Java-based backend through a REST interface. The backend, outlined in Section 2.2, is implemented using the Spring Framework and utilizes the OWL API [1] to create customized ontologies.

2.1. Graphical User Interface (GUI)

The graphical interface of OWL2Gen is designed for simplicity and efficiency, catering to users with varying levels of expertise to easily generate customized ontologies. As shown in Figure 1, the main interface features a configuration panel that lists all OWL 2 constructs, grouped into eight distinct categories according to the OWL 2 Quick Reference Guide[2]. This structure mirrors the one used in OntoBench. The categories are arranged within frames in the GUI, and predefined buttons allow users to quickly select presets, such as choosing all elements within a specific category.

Once the user selects the desired OWL 2 constructs, they can specify the quantity needed for each selected construct. After configuring the desired settings, the user can click the *"Generate"* button. At this point, OWL2Gen communicates with the backend via the REST API, where the user's configuration is processed and transformed into a fully functional ontology.

OWL2Gen supports various OWL serialization formats, making it adaptable to various applications and system requirements. Users can select their preferred serialization format from a drop-down menu, which includes all formats supported by the OWL API, which currently include Turtle, Manchester, Functional, OWL/XML, and RDF/XML.

2.2. Methodology

The methodology leverages existing OWL 2 DL TBoxes to generate ontologies, allowing users to either provide their own input ontologies or use the default ones provided. In this approach,

[1]https://owlapi.sourceforge.net/
[2]https://www.w3.org/TR/2012/REC-owl2-quick-reference-20121211/

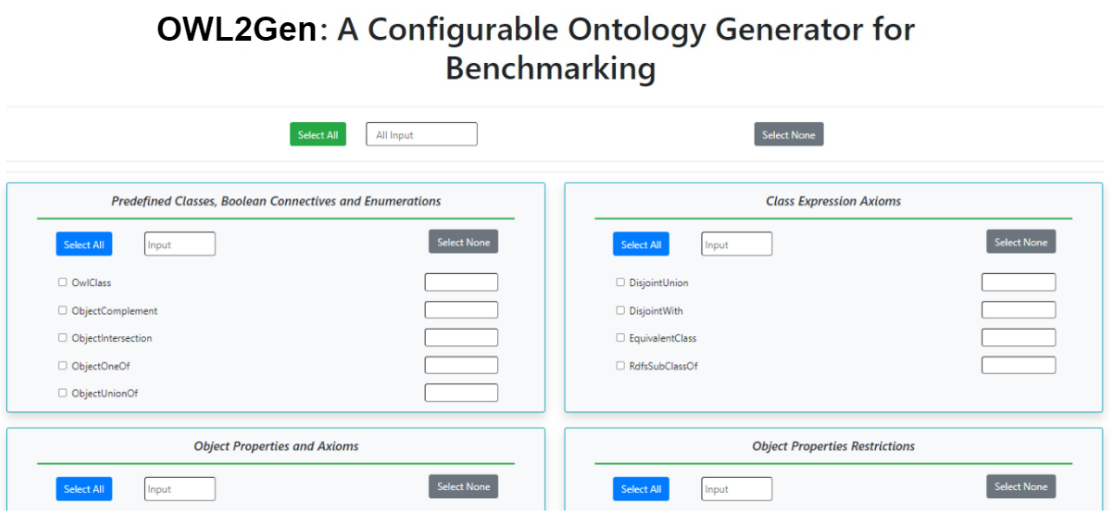

Figure 1: The configuration panel of OWL2Gen. Each construct includes a checkbox and an input field for specifying the count. Users can also select all constructs within a category or choose to select all the OWL 2 constructs from the top.

we first create separate buckets for each OWL 2 construct, all initially empty. Once the user selects the input base ontologies, axioms belonging to different constructs are sorted into the corresponding buckets. For example, axioms like `Faculty` $\sqsubseteq \exists$ `worksFor.College` would be placed in the *existential restriction* bucket. These axioms serve as the foundation for generating new ontologies. If the user selects *existential restriction*, axioms from this bucket will be picked for inclusion in the new ontology.

If the required count for a construct is less than or equal to the available axioms, they are used directly. However, if the required count exceeds the number of axioms in the bucket, new axioms are generated by replacing term(s) in the original axiom. For instance, introducing a new class `Faculty_1` can lead to generating an axiom like `Faculty_1` $\sqsubseteq \exists$`worksFor.College`. This approach ensures consistency and readability by aligning the naming conventions with the original vocabulary. The decision on which terms to replace is influenced by factors such as maximizing class reuse and ensuring sufficient connections between entities to avoid unnecessary new classes. Our generation pipeline leverages data structures that rank potential axioms based on the usage frequency of the entities involved in the axiom and the interconnections among them so far. These structures are consulted before generating new axioms, optimizing class and property reuse while maintaining complex connections. Additionally, we introduce some randomness into the process, such as determining whether `Faculty` and `Faculty_1` should both be subclasses of `Employee` or if they should be subclasses of `Employee` and `Employee_1`, respectively.

Since we rely on an existing input TBox, some axioms may include related hierarchical or domain/range axioms. For example, alongside `Faculty` $\sqsubseteq \exists$`worksFor.College`, additional axioms such as `Faculty` $\sqsubseteq$ `Employee`, `College` $\sqsubseteq$ `University`, and `University` $\sqsubseteq$ `Organization` might be relevant. To account for this, we provide users the option to retain these hierarchical and domain/range connections in the generated ontology, ensuring a more complete and coherent structure that better reflects real-world relationships and reasoning scenarios.

Furthermore, while reasoners may handle one set of axioms generated for certain constructors efficiently, a different set of axioms with the same constructors could cause computational blow-ups due to unforeseen interactions. To address this variability, we generate four distinct ontologies in each run. Given that each constructor's bucket contains multiple axioms, this approach enables the creation of varied ontologies. Further details on the approach and design choices are available in the documentation at https://github.com/kracr/owl2gen.

2.3. Use Case Scenarios

The OWL2Gen ontology generator presents many use cases that significantly enhance research and practical applications in ontology engineering. The key application lies in benchmarking various ontology reasoning systems, including conventional and emerging neuro-symbolic reasoners. Since, the generation algorithm leverages an input ontology, users can generate ontologies in two ways: first, by applying the same set of OWL 2 constructs to the same input ontology, and second, by using the same constructs across different input ontologies. This results in ontologies with varying structural characteristics, even with identical constructs. Researchers can compare these generated ontologies and observe how they exhibit different performance characteristics, highlighting the impact on reasoning efficiency. Additionally, users can generate ontologies by gradually increasing the count of individual constructs or combinations of constructs. This iterative process helps in systematically identifying bottlenecks in reasoning performance.

Further, the generator allows users to create ontologies that range from simple structures with fewer axioms to more complex designs with intricate relationships. This variation enables researchers to investigate specific cases where smaller ontologies, despite their simplicity, may challenge reasoning engines due to unexpected complexities in their design. Conversely, larger ontologies, which are often more structured, can demonstrate faster reasoning times, providing insights into how these systems handle scalability and efficiency.

Moreover, the configurable nature of OWL2Gen enables users to produce ontologies tailored to the unique requirements of neuro-symbolic reasoners, which integrate neural networks with symbolic reasoning. By generating ontologies that reflect the particular demands of neuro-symbolic tasks, users can explore how these systems respond to different structural designs and reasoning scenarios.

Additionally, OWL2Gen enhances the benchmarking of visualization tools by providing a rich variety of generated ontologies that can be used to assess and compare their effectiveness. By creating ontologies with diverse structures and complexities, researchers can evaluate how well different visualization platforms represent intricate relationships and semantics. This process not only helps identify the strengths and weaknesses of visualization tools but also drives improvements in their design. For beginners, OWL2Gen offers a user-friendly interface to experiment with various ontology configurations and understand their implications. In sensitive domains, it can simulate ontologies that meet specific regulatory requirements, enabling safe and compliant exploration of data relationships. This multifaceted utility positions OWL2Gen as a vital asset for academic research and practical applications across diverse fields.

3. Conclusion and Future Work

In this paper, we present our initial efforts to develop a configurable ontology generator that supports all OWL 2 language constructs. Currently, our tool enables the generation of scalable ontologies based on input TBoxes; however, it requires users to provide a complete TBox with all necessary constructs upfront, which limits flexibility for various applications. To address this limitation, we are working on a more general approach to enhance the tool's usability. Our vision is to create a configurable ontology generator that dynamically adapts to varying input requirements without needing a fully predefined TBox. We also plan to expand support for various OWL 2 profiles (EL, QL, RL) to ensure compliance with syntactic and semantic constraints. Additionally, we aim to support incremental ontology building, allowing users to start with basic constructs and progressively add complexity based on insights from an integrated visualization tool, which will provide users with valuable insights into the generated ontology's complexity.

References

[1] P. Hitzler, M. Krötzsch, B. Parsia, P. F. Patel-Schneider, S. Rudolph, et al., Owl 2 web ontology language primer, W3C recommendation 27 (2009) 123.

[2] M. Krötzsch, OWL 2 profiles: An introduction to lightweight ontology languages, in: T. Eiter, T. Krennwallner (Eds.), Reasoning Web. Semantic Technologies for Advanced Query Answering - 8th International Summer School 2012, Vienna, Austria, September 3-8, 2012. Proceedings, volume 7487 of *Lecture Notes in Computer Science*, Springer, 2012, pp. 112–183. URL: https://doi.org/10.1007/978-3-642-33158-9_4. doi:10.1007/978-3-642-33158-9_4.

[3] Y. Guo, Z. Pan, J. Heflin, LUBM: A Benchmark for OWL Knowledge Base Systems, Journal of Web Semantics. 3 (2005) 158–182.

[4] L. Ma, Y. Yang, G. Qiu, Z .and Xie, Y. Pan, S. Liu, Towards a Complete OWL Ontology Benchmark, in: The Semantic Web: Research and Applications, Springer Berlin Heidelberg, 2006, pp. 125–139.

[5] G. Singh, S. Bhatia, R. Mutharaju, OWL2Bench: A Benchmark for OWL 2 Reasoners, in: The Semantic Web - ISWC 2020 - 19th International Semantic Web Conference, ISWC 2020, Lecture Notes in Computer Science, Springer, 2020.

[6] V. Link, S. Lohmann, H. F., OntoBench: Generating Custom OWL 2 Benchmark Ontologies, in: International Semantic Web Conference, 2016, pp. 122–130.

[7] N. Hubert, P. Monnin, M. d'Aquin, D. Monticolo, A. Brun, Pygraft: Configurable generation of synthetic schemas and knowledge graphs at your fingertips, in: The Semantic Web - 21st International Conference, ESWC 2024, Hersonissos, Crete, Greece, May 26-30, 2024, Proceedings, Part II, volume 14665 of *Lecture Notes in Computer Science*, Springer, 2024, pp. 3–20.