VEL: A Formally Verified Reasoner for $\mathscr{EL}$++ Description Logic

Atalay Mert Ileri[1], Hande Küçük McGinty[1,*]

[1]Kansas State University, Department of Computer Science, Koncordant Lab, Manhattan, KS 66503

Abstract

Over the past two decades, the Web Ontology Language (OWL) has been instrumental in advancing the development of ontologies and knowledge graphs, providing a structured framework that enhances the semantic integration of data. However, the reliability of deductive reasoning within these systems remains challenging, as evidenced by inconsistencies among popular reasoners in competitions. This evidence underscores the limitations of current testing-based methodologies, particularly in high-stakes domains such as healthcare. To mitigate these issues, in this study, we have developed VEL, a formally verified $\mathscr{EL}$++ reasoner equipped with machine-checkable correctness proofs that ensure the validity of outputs across all possible inputs. This formalization, based on the algorithm of Baader et al.[1], has been transformed into executable OCaml code using the Coq proof assistant's extraction capabilities. In addition to producing a correct implementation, our work uncovered two errors in the published completeness proof that required a modification to the original algorithm.

Keywords

verified reasoner, reasoning, formal verification, knowledge graph, trustworthy AI

1. Introduction

Knowledge graphs and ontologies can integrate diverse data sources semantically rigorously, adhering to well-established W3C standards [2, 3]. They bridge human conceptualization and machine understanding and provide a robust standard for recording provenance [4], making results drawn from them inherently explainable and interpretable. However, continuous efforts for a more reliable reasoning need to be established by the semantic web community [5]. Reasoners are a group of software that are used to reach logical conclusions using knowledge graphs and ontologies. However, like any other software system, the reasoner implementations are susceptible to software bugs that undermine their correctness [6, 7, 8, 9, 10, 11, 12, 13]. A competition from 2015 for reasoners showed that some of the widely used reasoners contain bugs since the results of the different reasoners did not agree [14]. This result, combined with the increased usage of AI in critical domains like healthcare, shows that testing-based methodologies are insufficient to provide strong correctness guarantees.

To address this problem, in this paper, we introduce VEL, a formally verified $\mathscr{EL}$++ reasoner with machine-checkable correctness proofs. Our proofs ensure the correctness of the output

Posters, Demos, and Industry Tracks at ISWC 2024, November 13–15, 2024, Baltimore, USA

*Corresponding author.

atalay@ksu.edu (A. M. Ileri); hande@ksu.edu (H. K. McGinty)

https://www.koncordantlab.com/ (A. M. Ileri); https://www.koncordantlab.com/ (H. K. McGinty)

0009-0002-3610-4963 (A. M. Ileri); 0000-0001-7116-9338 (H. K. McGinty)

for each possible input. We based our formalization on the algorithm of Baader et al. [1] and obtained an executable OCaml code through the extraction functionality of Coq proof assistant [15].

2. Related Work

Baader et al. use automated theorem proving to certify the results of the ELK reasoner for OWL2 EL [16]. They implemented an algorithm that extracts a proof certificate from an ELK result and checks its correctness using LFSC proof checker. This approach incurs time and memory overheads due to certificate generation and proof checking at runtime and lacks support for required extensions such as concrete domains. In contrast, our implementations do not incur any runtime overhead since the correctness proofs of our implementations are statically checked. Eliminating this overhead will make our reasoners more scalable compared to validation-based approaches.

Hidalgo-Doblado et al. formalize $\mathscr{ALC}$ description logic and implement a formally verified tableau-based reasoner in PVS [17]. Although it is a well-known description logic, $\mathscr{ALC}$ does not correspond to any OWL2 profile and lacks the support for datatypes and other extensions. These limitations restrict the usability of their implementation in projects. Given these limitations and lack of support, our work is guided by OWL2 and supports necessary extensions to ensure our implementations' widespread usability.

3. Formalization of $\mathscr{EL}$++

The first step in implementing a verified reasoner is formalizing its logic, which consists of formalizing the syntax, such as concept descriptions and constraints, as well as the semantics of the logic. We formalized the logic parameterized by its primitives, such as names, to ensure its reusability in other projects. The parameterized implementation allowed us to obtain theorems agnostic to the encoding used to construct constraints, increasing the interoperability of the resulting implementation with various systems.

One challenging aspect of formalizing the language was formalizing concrete domains. We formalized concrete domains as a Coq record with six definitions: domain as a nonempty set of concrete domain elements, predicates as a set of predicate names, predicate arities as a partial function from predicate names to natural numbers, `apply` function that computes the application of a predicate, and satisfiability and implication functions that compute their respective properties between conjunctions of predicate expressions.

We formalized the provided model-theoretic semantics for $\mathscr{EL}$++ . We first defined what we call *a base interpretation* which maps each name to its corresponding structure in the model. Then, we defined the interpretation function that recursively constructs the interpretation of a concept description for a given base interpretation.

4. Formalization of Normalization and Classification

Since normalization and classification use the same structure, namely, identifying a candidate and applying the appropriate rule, we followed the same pattern in their formalization. We formalized each rule in two parts: a predicate that encodes the condition and the application. In our implementations, we use a higher-order function to identify a candidate that satisfies a predicate, then apply the corresponding rule until no more candidates exist.

Defining a recursive function in Coq requires proving it is well-founded, i.e., it will terminate for all inputs. One way to do that is by providing a measure and proving that it decreases after each recursive call. A measure based on the number of rule applications left was sufficient for most rules.

We had to make one modification to the normalization rule that applies to constraints of the form $C \sqcap D \sqsubseteq E$ where C or D are complex concept descriptions. The termination measure does not decrease when the rule is applied to a constraint where both C and D are complex due to generating a new candidate. We circumvented this problem by applying the rule twice when both concept descriptions are complex.

We implemented strings and rational numbers as examples of concrete domains. Due to their complexity, we treated satisfiability and implication in those domains axiomatically and implemented them as unverified OCaml code. We used Coq's extraction functionality to obtain an executable OCaml code of the reasoner.

5. Proving Correctness

Completeness proof was more challenging to mechanize than soundness proof. It required us to define a new invariant, fix errors in the original proof, and employ non-constructive reasoning.

Point 2 of Claim 1 in the original completeness proof involved a property that holds for the final classification but does not necessarily hold in every intermediate step. To mechanize the proof, we defined a tree structure that encodes possible sequences of rule applications that lead to the current state. This structure allowed us to mimic doing induction over selected rule applications by doing induction over the trees.

5.1. Errors in Completeness Proofs

Our mechanization revealed two major errors in pen-and-paper completeness proofs. The first one is the non-transitivity of a relation assumed to be transitive. The second one is the under-specification of a solution for concrete domain predicates. We explain each problem and our solutions below.

5.1.1. Non-transitivity

Completeness proofs in Baader et. al. heavily rely on a relation defined with respect to a concept name A denoted by $\sim_A$. During our formalization, we discovered that $\sim_A$ is not transitive. The problem stemmed from the fact that in some models of the knowledge base, the interpretation of A can be empty. We fixed this error by introducing A-extensions to the algorithm that enforce

nonemptiness while preserving subsumption. A-extension adds $\{t\} \sqsubseteq \exists r_t.A$ constraint to the CBox, where t and r_t are fresh individual and role names, respectively.

5.1.2. Under-specification

We identified another error in the statement of a lemma, originally called Claim 2, used in constructing a counterexample model. Original lemma states that "For each concrete domain and $\sim_A$ equivalence class, there exists a solution such that [...]". However, this was too weak to imply the desired properties. We fixed this error by reordering the quantification to "For each $\sim_A$ equivalence class, there exists a solution such that, for each concrete domain [...]" and further restricting the behavior of the solution to the feature names that appear in the classification of the equivalence class.

6. Evaluation

We evaluated our work in two axes. We measured the required proof effort by comparing the number of lines of proof per line of implementation. Our 165 definitions consist of 1511 cloc. Our 387 theorems consist of 16977 cloc, giving us 11.3x proof overhead.

To evaluate the performance of our implementation, we measured run times on randomly generated knowledge bases with fixed set of names. For concept inclusion tests, we kept the number of role inclusion axioms at 10, and for role inclusion tests, we set the number of concept inclusions to 20. Our algorithm's runtimes are 4s for 20, 278s for 30, and 192s for 40 concept inclusions; and 463s for 20, 6s for 30, and 2732s for 40 role inclusions. Our results indicates that the structure of a knowledge base more impactful on the performance than its size. We could not test our performance against the existing reasoners since interfacing with OWL is not implemented yet.

7. Conclusion

The Semantic Web, over the past three decades, helped enhance a wide range of applications. However, the varying sophistication of SW systems' reasoning abilities highlights the need for reliable reasoning algorithms, particularly in the context of the growing interest in Neurosymbolic AI and Generative AI approaches that seek to utilize Semantic Web layers.

In an attempt to address this gap, our work lays the foundation for formally verified reasoners for Neurosymbolic AI applications. We believe that, in the cases of high-stakes application domains, such as health, finance, and security, the benefit of increased trustworthiness outweighs the increased development effort.

As future work, we are planning to integrate our reasoner into Protègè ontology editor [18] by creating a plugin to provide access to our reasoner for community use. We believe that having an easy access to verified reasoners will contribute to the adoption of formal methods in trustworthy AI. The second direction we will explore is reducing the trusted computing base by implementing a verified parser that will convert an OWL file to a VEL knowledge base.

References

[1] F. Baader, S. Brandt, C. Lutz, Pushing the el envelope, in: Proceedings of the 19th International Joint Conference on Artificial Intelligence, IJCAI'05, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2005, p. 364–369.

[2] R. Cyganiak, D. Wood, M. Lanthaler (Eds.), RDF 1.1 Concepts and Abstract Syntax, W3C Recommendation 25 February 2014, 2014. Available from http://www.w3.org/TR/rdf11-concepts/.

[3] H. Knublauch, D. Kontokostas (Eds.), Shapes Constraint Language (SHACL), W3C Recommendation 20 July 2017, 2017. Https://www.w3.org/TR/shacl/.

[4] S. Sahoo, D. McGuinness, T. Lebo, PROV-O: The PROV Ontology, W3C Recommendation, W3C, 2013. Http://www.w3.org/TR/2013/REC-prov-o-20130430/.

[5] H. K. McGinty, KNowledge Acquisition and Representation Methodology (KNARM) and Its Applications, Ph.D. thesis, University of Miami, 2018.

[6] rusdobr, [bug] reasoner fails to validate when integer is present in ontology iri, https://github.com/stardog-union/pellet/issues/49, 2024. Accessed: 2024-08-29.

[7] patrickwestphal, Got decimal type for integer value inferred via hasvalue restriction, https://github.com/stardog-union/pellet/issues/38, 2017. Accessed: 2024-08-29.

[8] tobias hammerschmidt, Literal.isdifferent can return wrong result for numeric literals, https://github.com/stardog-union/pellet/issues/34, 2016. Accessed: 2024-08-29.

[9] Christian-D, Indexoutofboundsexception in betanode.java for rules with multiple atoms like p(?x,?x), p(?y, ?y), https://github.com/stardog-union/pellet/issues/4, 2014. Accessed: 2024-08-29.

[10] leechuck2@gmail.com, Nullpointer exception when classifying fma, https://github.com/liveontologies/elk-reasoner/issues/35, 2015. Accessed: 2024-08-29.

[11] naditina, Nullpointer while classifying the el fragment of galen, https://github.com/liveontologies/elk-reasoner/issues/38, 2016. Accessed: 2024-08-29.

[12] JieyingChenChen, java.lang.outofmemoryerror: unable to create new native thread, https://github.com/liveontologies/elk-reasoner/issues/36, 2016. Accessed: 2024-08-29.

[13] balhoff, 'direct' subclasses are not included when 'all' subclasses are requested for anonymous class expressions, https://github.com/liveontologies/elk-reasoner/issues/70, 2024. Accessed: 2024-08-29.

[14] B. Parsia, N. Matentzoglu, R. S. Gonçalves, B. Glimm, A. Steigmiller, The owl reasoner evaluation (ore) 2015 competition report, Journal of Automated Reasoning 59 (2017) 455 – 482. URL: https://api.semanticscholar.org/CorpusID:9446588.

[15] The Coq Development Team, The Coq Proof Assistant, version 8.19.2, 2024. URL: https://doi.org/10.5281/zenodo.1003420. doi:10.5281/zenodo.1003420.

[16] F. Baader, P. Koopmann, C. Tinelli, First results on how to certify subsumptions computed by the el reasoner elk using the logical framework with side conditions, Description Logics (2020). URL: https://api.semanticscholar.org/CorpusID:221717879.

[17] M. J. Hidalgo-Doblado, J. A. Alonso-Jiménez, J. Borrego-Díaz, F. J. Martín-Mateos, J. L. Ruiz-Reina, Formally verified tableau-based reasoners for a description logic, J. Autom. Reason. 52 (2014) 331–360. URL: https://doi.org/10.1007/s10817-013-9291-8. doi:10.1007/s10817-013-9291-8.

[18] M. A. Musen, The protégé project: a look back and a look forward, AI Matters 1 (2015) 4–12. URL: https://doi.org/10.1145/2757001.2757003. doi:10.1145/2757001.2757003.