Testing Description Logic Reasoners (Extended Abstract)

Tobias John[1], Einar Broch Johnsen[1], Eduard Kamburjan[1,2] and Dominic Steinhöfel[3]

[1]University of Oslo, Norway
[2]IT University of Copenhagen, Denmark
[3]CISPA Helmholtz Center for Information Security, Germany

Abstract

Ontologies rely on a vast ecosystem of software tools, especially the description logic reasoners. Yet, tool developers and maintainers have little support to ensure tool reliability. This extended abstract reports on two testing studies on EL reasoners, summarizing results from two approaches presented at ISSRE'24 [1], ESWC'25 [2] and preliminary results building on this work [3]. We demonstrate how recent advances in test case generation for highly structured and constrained inputs can support software developers in the semantic web field. We develop input generators for the OWL EL profile, and report on 23 bugs we found in HermiT, ELK and Pellet, as well as the EL profile checker of the OWL-API, all using *automated testing*.

Keywords

Software Testing, Software Quality, Reasoners

1. Introduction

Motivation. Ontologies are used in business-critical applications and are supported by a rich ecosystem of standards, technologies and software tools [4]. Nevertheless, quality considerations are mostly concerned with data or ontology quality [5, 6], and rarely consider the software tools that underlie the application. However, only few investigations into reliability of the software ecosystem surrounding ontologies have been conducted, which diminishes confidence in their results—confidence that is critical especially if ontologies are used to improve confidence in an overall application, e.g., to counteract gaps or hallucinations in neuro-symbolic AI [7, 8]. In the related field of SMT-solvers, testing is an active area of research [9, 10, 11, 12]. Testing tools have found and documented thousands of bugs in mature SMT-solvers. In this work we present the first results of applying testing techniques to EL reasoners, published at ESWC'25 [2], and extended results based on work published at ISSRE'24 [1] and submitted to a journal [3].

Overview. We tested the solvers HermiT (v.1.4.5.51) [13], Pellet [14] (using its Java binding Openllet, v.2.6.5), and ELK (v.0.6.0) [15]. These reasoners were selected as they are (a) openly available and (b) preinstalled with Protégé [16], which is still the most widely used ontology modelling tool. Only ELK is under active maintenance. In the following, when we refer to the reasoners, we refer to the respective version listed above.

There were two testing campaigns, both based on differential testing [17]: We generate (semi)random ontologies, give them as inputs to all three reasoners, and consider it an *anomaly* if one reasoner disagrees with the results of the others, or throws an exception. We tested two capabilites: (1) We checked for consistency of the input ontology, and (2) we checked for the inference of all axioms of the following types: SubClass, DisjointClasses, EquivalentClasses, ClassAssertion, PropertyAssertion, SubObjectProp, SubDataProp, EquivalentObjectProp, EquivalentDataProp, as well as characteristics of object and data properties, e.g. functionaliy or symmetry. As the ELK reasoner does not support the

DL 2025: 38th International Workshop on Description Logics, September 3–6, 2025, Opole, Poland
tobiajoh@ifi.uio.no (T. John); einar@ifi.uio.no (E. B. Johnsen); eduard.kamburjan@itu.dk (E. Kamburjan); steinhoefel@cispa.de (D. Steinhöfel)
0000-0001-5855-6632 (T. John); 0000-0001-5382-3949 (E. B. Johnsen); 0000-0002-0996-2543 (E. Kamburjan); 0000-0003-4439-7129 (D. Steinhöfel)

inference of all of these types of axioms, we only checked if the inferred axioms of ELK are a subset of the axioms inferred by the other reasoners.

Both campaigns also considered other tools, we only report on those results related to the reasoner. The full description of the bugs can be found in the original papers [3, 1, 2].

2. Language-Based Fuzzing Campaign

Approach. In the first campaign [2], we used language-based fuzzing and the ISLa tool [18]: Language-based fuzzing uses an input grammar and generates random terms that are accepted by it, possibly constrained by additional conditions on the term.

The generator produces OWL files in functional syntax, which is defined in the specification of the OWL EL profile [19]. However, we needed to make some adjustments to the grammar. Firstly, we restrict the grammar to use four different names per category: classes, individuals, simple object properties, potentially non-simple object properties, data properties, predefined data types, custom data types, language tags, strings, annotation properties and annotation values, respectively. This ensures that the names are used multiple times in the ontology. Secondly, the grammar is modified to encode some of the constraints on the EL profile that are not contained in the original grammar, e.g., the restrictions on complex roles. We do not encode all of the constraints, namely that cyclic definitions of properties are forbidden and the *global restrictions on the EL profile* [19]. Thus, we also generated test cases that are not in OWL EL, which the OWL-API profile checker and ELK are expected to reject.

Results. The main test run was performed for 10 hours, producing and evaluating 1,557 test cases. Out of those, 516 test cases contain at least one anomaly. The test cases are manually classified and we examined at least one test case in each class.

Some classes turn out to not be bugs of the reasoners but rather mirror specific reasoning limitations of the reasoners. We do not examine unclassified anomalies with very different behavior of the tools as this indicates the occurrence of several anomalies at the same time, which we already examine individually in other classes. While investigating the anomalies, the input files are simplified, i.e. axioms and parts of axioms that do not affect the result are removed. Classes, individuals and properties are renamed to make the files easier to understand. This last step also revealed one bug.

Overall, this testing methodology revealed 14 previously unknown bugs in the reasoners and 1 related bug in the OWL-API. Table 1 provides an overview of these bugs. We found bugs in all systems that we tested. We reported the bugs in the repositories of the tested tools, except for HermiT where no bug tracker is publicly available. The detailed reports for those bugs (as well as all other bugs) can be found in the supplementary material of the original publication [23]. One interesting observation is that most of the found bugs are *logical bugs*, i.e. bugs that do not lead to any exception or warning but the computed result is incorrect. Logical bugs are particularly important to find, as there is no indication of their presence for users of the tools. To illustrate the kinds of bugs we found, we discuss a few different classes of bugs using examples.

Parsing. The OWL EL profile allows language tags, but the OWL-API parser combines strings and attached language tags to object of type `rdfs:langString`. The EL-profile checker of the OWL-API subsequently rejects the ontology because this data type is forbidden, effectively forbidding the use of language tags.

Consistency. Four bugs are cases where the consistency of the ontology is not correctly assessed. We found three such bugs in Pellet/Openllet and one in ELK. The latter (*E1* in Table 1) is shown in Figure 1. The ontology is wrongly classified as inconsistent by ELK: ELK treats the two strings `"s1"@fr` and `"s1"@en` as the same entity, even though they must be treated as different literals. This leads to the incorrect inference that `:B` is an empty class and, due to the class assertion **ClassAssertion**`(:B :a)`, that the ontology is inconsistent.

Table 1
Bugs from the language-based campaign. **Issue** refers to the corresponding issue tracker [20, 21, 22]. Legend: S = Soundness, SC = Soundness (consistency), SI = Soundness (inference), C = Completeness, D = Documentation, E = Exception.

Tool	ID	Issue	Type	Summary
Pellet	*O1*	85	SC	Equivalence to bottom property is missing when range of property is empty
	O2	87	SI	Incorrectly infers asymmetry of property from a single triple
	O3	88	SI	Incorrect equivalence with bottom class
	O4	89	S	Renaming of variables leads to change in consistency check
	O5	90	SC	Ignores language tags when checking for equivalence of literals
	O6	91	E	Null-pointer exception on invocation
	O7	92	SI	Incorrectly infers irreflexivity of property from single triple
ELK	*E1*	71	SC	Ignores language tags when considering equivalence of literals.
HermiT	*H1*	—	C	Missing `owl:Thing` class assertion of decl. individ. when no class is declared
	H2	—	D	Warning about malformed input missing and computed result has SI issue
	H3	—	C	Functionality of property missing when property range is a singleton
	H4	—	SI	Incorrect sub-object-property relation caused by unrelated subclass axiom
	H5	—	C	Data-property assertion missing when singleton in equivalent-classes axiom
	H6	—	E	Exception when using only `rdfs:Literal` in data-type intersection
OWL-API	*A1*	1158	S	Forbidden data type is wrongly detected because of incorrect parsing of string with language tag as `rdfs:langString`
	A2	1159	D	Too few arguments are detected because argument-list is interpreted as set
	A3	1160	S	Defining new data type is incorrectly marked as violation (only use forbidden)

```
Prefix(:=<http://www.example.org/reasonerTester#>)
Ontology (
    Declaration(Class(:B))   Declaration(Class(:A))
    Declaration(DataProperty(:dr))   Declaration(NamedIndividual(:a))
    EquivalentClasses( DataHasValue(:dr "s1"@fr)   :A   :B )
    DisjointClasses(   DataHasValue(:dr "s1"@en)   :A )
    ClassAssertion(:B :a))
```

Figure 1: Consistency bug in ELK (bug *E1*)

```
Prefix(:=<http://www.example.org/reasonerTester#>)
Ontology (
    Declaration(DataProperty(:dp))   Declaration(NamedIndividual(:a))
    EquivalentClasses( ObjectOneOf(:a)   DataHasValue(:dp "data") ))
```

Figure 2: Inference bug: completeness bug in HermiT (bug *H5*)

Soundness of Inference. Five of the bugs are cases where the reasoners infer axioms that are not entailed by the ontology. These bugs were found in HermiT and Pellet/Openllet. For one such bug (*O4*), the ontology contains only one axiom **EquivalentClasses**`(ObjectHasSelf(:qsim) ObjectOneOf(:d) :C))` and declares an individual `:a` that is not mentioned in the axiom. Nevertheless, it is inferred that the axiom **ClassAssertion**`(:C :a)` is entailed by the ontology. Interestingly, changing the name of the declared class, the individuals or the properties leads to a correct inference, i.e., the bug disappears. We see this puzzling behavior of Pellet/Openllet for several different inputs.

Completeness of Inference. Three of the bugs are cases where the reasoners do not infer an axiom. All these bugs were found in HermiT. Figure 2 illustrates *H5*. The ontology defines a data property `:dp` and an individual `:a`. Furthermore, two classes are the same: (i) the class containing only the individual `:a` and (ii) the class of individuals occurring as the subject in an assertion with the data property `:dp` and the object `"data"`. One can therefore infer the assertion **DataPropertyAssertion**`(:dp :a "data")` from the ontology. However, HermiT does not infer the axiom when asked to compute all inferred assertions.

Table 2
Bugs from the mutation-based campaign. Legend is given in Table 1.

Tool	ID	Issue	Type	Summary
Pellet	*P1*	94	E	Exception when pre-computating class hierarchy; occurs non-deterministically
	P2	93	C	A sub-class axiom is missing from inferred class hierarchy; combination of reflexive property and existential quantification
	P3	97	E	Exception when doing pre-computation for class hierarchy
	P4	95	E	Exception when doing pre-computation for class hierarchy
	P5	96	C	A sub-class axiom is missing from inferred class hierarchy because sub-typing of different string data types is not considered
HermiT	*H0*	—	E	Exception when reasoner is initiated if ontology contains the axiom $\bot \sqsubseteq \top$

3. Mutation-based Campaign

Approach. In the second campaign [3], we used mutation-based fuzzing. Mutation-based fuzzing uses seed inputs and generates random terms by applying mutation operators (such as removing, adding or exchanging a subterm) on the seed repeatedly.

To get representative ontologies as seeds, we collected ontologies from the latest OWL Reasoner Competition [24]. We selected all ontologies in the EL-profile and then filtered to only use the ontologies that are smaller than 1MB in file size. This resulted in 307 ontologies that we used as seeds. Again, we performed a test run for 10h, during which we generated 1991 test cases, of which 502 test runs resulted in the occurrence of an anomaly. The used mutation operators are selected such that they address all allowed features of the EL-profile. The full list of all 55 used mutation operators can be found in the original publication [3]. They target the TBox as well as the ABox. The used tool is available under https://github.com/smolang/RDFMutate [25].

Results. We found 6 previously unknown bugs in the OWL EL reasoners that we tested. One of the bugs (*P1*) only occurs non-deterministically, i.e., not every time the reasoner is called with the test input. Such bugs are particularly hard to describe. Two of the bugs are completeness bugs, i.e., the reasoner misses a subclass relation that should be contained in the class hierarchy. These bugs are especially important to find as there is no warning for the user that the computed result of the tool is incorrect.

4. Conclusion

Software reliability, or software quality in general, is an underestimated and underappreciated factor when it comes to ontologies, yet reliability is critical for the acceptance of this technology. The reported work shows that automated testing can be applied to DL-reasoners and, subsequently, should be applied in the future. In particular, it demonstrates that language-based tools for input generation have reached a stage where they can be set up for OWL with little effort. For future work, it remains to investigate further testing techniques, in particular gray-box fuzzing [26] and other approaches that are building on *coverage* to estimate how much of the program or input space has been already tested. It also remains to investigate why language-based fuzzing was more effective than mutation-based fuzzing.

Declaration on Generative AI

The author(s) have not employed any Generative AI tools.

References

[1] T. John, E. B. Johnsen, E. Kamburjan, Mutation-based integration testing of knowledge graph applications, in: ISSRE, ACM, 2024, pp. 475–486. doi:10.1109/ISSRE62328.2024.00052.

[2] T. John, E. B. Johnsen, E. Kamburjan, D. Steinhöfel, Language-based testing for knowledge graphs, in: ESWC, volume 15719 of *Lecture Notes in Computer Science*, Springer, 2025. doi:10.1007/978-3-031-94578-6_2.

[3] T. John, E. B. Johnsen, E. Kamburjan, Mutation-based testing of knowledge graph applications, 2025. Under submission.

[4] C. Gutierrez, J. F. Sequeda, Knowledge graphs, Commun. ACM 64 (2021) 96–104. doi:10.1145/3418294.

[5] D. Vrandecic, Ontology evaluation, in: Handbook on Ontologies, International Handbooks on Information Systems, Springer, 2009, pp. 293–313. doi:10.1007/978-3-540-92673-3_13.

[6] B. Xue, L. Zou, Knowledge graph quality management: A comprehensive survey, IEEE Trans. Knowl. Data Eng. 35 (2023) 4969–4988. doi:10.1109/TKDE.2022.3150080.

[7] K. Sun, Y. E. Xu, H. Zha, Y. Liu, X. L. Dong, Head-to-tail: How knowledgeable are large language models (LLMs)? A.K.A. will LLMs replace knowledge graphs?, in: Conference of the North American Chapter of the Association for Computational Linguistics NAACL-HLT, Association for Computational Linguistics, 2024, pp. 311–325. doi:10.18653/V1/2024.NAACL-LONG.18.

[8] X. L. Dong, Generations of knowledge graphs: The crazy ideas and the business impact, Proc. VLDB Endow. 16 (2023) 4130–4137. doi:10.14778/3611540.3611636.

[9] R. Brummayer, A. Biere, Fuzzing and delta-debugging SMT solvers, in: SMT, 2009, pp. 1–5. doi:10.1145/1670412.1670413.

[10] D. Winterer, Z. Su, Validating SMT solvers for correctness and performance via grammar-based enumeration, Proc. ACM Program. Lang. 8 (2024) 2378–2401. doi:10.1145/3689795.

[11] M. N. Mansur, M. Christakis, V. Wüstholz, F. Zhang, Detecting critical bugs in SMT solvers using blackbox mutational fuzzing, in: ESEC/FSE, ACM, 2020, pp. 701–712. doi:10.1145/3368089.3409763.

[12] D. Winterer, C. Zhang, Z. Su, Validating SMT solvers via semantic fusion, in: International Conference on Programming Language Design and Implementation PLDI, ACM, 2020, pp. 718–730. doi:10.1145/3385412.3385985.

[13] B. Glimm, I. Horrocks, B. Motik, G. Stoilos, Z. Wang, HermiT: An OWL 2 reasoner, J. Autom. Reason. 53 (2014) 245–269. doi:10.1007/S10817-014-9305-1.

[14] E. Sirin, B. Parsia, B. C. Grau, A. Kalyanpur, Y. Katz, Pellet: A practical OWL-DL reasoner, J. Web Semant. 5 (2007) 51–53007. doi:10.1016/J.WEBSEM.2007.03.004.

[15] Y. Kazakov, M. Krötzsch, F. Simancik, The incredible ELK - from polynomial procedures to efficient reasoning with EL ontologies, J. Autom. Reason. 53 (2014) 1–61. doi:10.1007/S10817-013-9296-3.

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

[17] D. R. Slutz, Massive stochastic testing of SQL, in: International Conference on Very Large Data Bases (VLDB), volume 98, ACM, 1998, pp. 618–622. URL: http://www.vldb.org/conf/1998/p618.pdf.

[18] D. Steinhöfel, A. Zeller, Input invariants, in: ESEC/FSE, ACM, 2022, pp. 583–594. doi:10.1145/3540250.3549139.

[19] I. Horrocks, Z. Wu, B. C. Grau, A. Fokoue, B. Motik, OWL 2 Web Ontology Language Profiles (Second Edition), W3C Recommendation, W3C, 2012. URL: https://www.w3.org/TR/2012/REC-owl2-profiles-20121211.

[20] Openllet development team, Openllet, issue tracker, 2024. URL: https://github.com/Galigator/openllet/issues.

[21] ELK development team, ELK reasoner, issue tracker, 2024. URL: https://github.com/liveontologies/elk-reasoner/issues.

[22] OWL-API development team, OWL-API, issue tracker, 2024. URL: https://github.com/owlcs/owlapi/issues.

[23] T. John, E. B. Johnsen, E. Kamburjan, D. Steinhöfel, Supplementary material for paper "Language-based testing for knowledge graphs", Zenodo, 2025. doi:10.5281/zenodo.14512591.
[24] B. Parsia, N. Matentzoglu, R. S. Gonçalves, B. Glimm, A. Steigmiller, The OWL reasoner evaluation (ORE) 2015 competition report, J. Autom. Reason. 59 (2017) 455–482. doi:10.1007/S10817-017-9406-8.
[25] T. John, E. B. Johnsen, E. Kamburjan, RDFMutate: Mutation-based generation of knowledge graphs, in: ISWC, 2025. Accepted for publication.
[26] M. Böhme, V. Pham, M. Nguyen, A. Roychoudhury, Directed greybox fuzzing, in: CCS, ACM, 2017, pp. 2329–2344. doi:10.1145/3133956.3134020.